AD-A264 730

(2)

RL-TR-92-317
Final Technical Report
December 1992

ROME LABORATORY

PROCEDURES FOR APPLYING ADA QUALITY PREDICTION MODELS

The MITRE Corporation

D.D. Murphy, W.M. Thomas, W.M. Evanco, W.W. Agresti

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-92-317 has been reviewed and is approved for publication.

APPROVED: Andrew J Chruscicki

ANDREW J. CHRUSCICKI
Project Engineer

FOR THE COMMANDER John A Graniero

JOHN A. GRANIERO
Chief Scientist for C3

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1992	Final Oct 91 - Sep 92

4. TITLE AND SUBTITLE

PROCEDURES FOR APPLYING ADA QUALITY PREDICTION MODELS

5. FUNDING NUMBERS

C - F19628-89-C-0001
PE - 62702F
PR - MOIE
TA - [illegible]
WU - [illegible]

6. AUTHOR(S)

D. D. Murphy, W. M. Thomas, W. M. Evanco, W. W. Agresti

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The MITRE Corporation
Washington C3 Center
7525 Colshire Drive
McLean VA 22102-3481

8. PERFORMING ORGANIZATION REPORT NUMBER

MTR 92 W0000165

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory (C3CB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-92-317

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Andrew J. Chruscicki/C3CB/(315) 330-[illegible]

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Procedures for applying Ada software quality prediction models for purposes of model validation are described. The multivariate regression models predict metrics related to software reliability, maintainability, and flexibility. The procedures include the use of an Ada source code analysis tool and the Statistical Analysis System (SAS) to extract data from Ada source code and create data sets containing quantities needed for the models.

14. SUBJECT TERMS

Software Quality, Ada Reliability, Maintainability, Flexibility

15. NUMBER OF PAGES 84

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

EXECUTIVE SUMMARY

This report describes procedures for applying Ada software quality prediction models for purposes of model validation. The multivariate regression models were developed under the Mission Oriented Investigation and Experimentation (MOIE) program of The MITRE Corporation. The models predict metrics related to software reliability, maintainability, and flexibility. The procedures include the use of an Ada source code analysis tool and the Statistical Analysis System (SAS) to extract data from Ada source code and create data sets containing quantities needed for the models.

The quality prediction models have been developed in a research setting, based on software project data. The models are at a stage of development where they are ready for validation on additional software projects to refine the coefficients of the models. Validation of the models on diverse software projects will increase the confidence in subsequent application of the models.

Readers are cautioned that the models were developed by analyzing source code and data from a particular set of Ada projects. The models in this report should not be expected to be universally applicable regardless of the size and nature of the project. Indeed, understanding the range of applicability of the models is part of the validation process, which the procedures in this report are intended to facilitate.

This report is intended to support individuals who want to validate the models by applying them to Ada projects. The starting point for someone to use this report is the availability of Ada source code and an interest in obtaining a static analysis of the code or applying the quality prediction models. The models are of the form,

$$q = f(a_i * X_i)$$

where q is a quality factor to be predicted; a_i are coefficients resulting from the MOIE research and given in this report; and X_i are calculated quantities whose values depend on the Ada source code for the project. The values of X_i need to determined, so they can be combined with the coeffficients a_i to produce the predicted quality factor. Validation involves comparing the predicted values to actual data as they become available on projects.

The models are based on static features of the Ada code, such as counts of declarations imported and exported across library units. To extract these data, a software tool, the Ada Source code Analyzer Program (ASAP), is used. An additional product of ASAP is the generation of a Project Summary Report, providing a profile of the source code. The extracted data proceed through several stages of processing before they are transformed into the X_i values needed for the quality prediction models. Because several processing steps are involved, an organization of directories and files has been established and described in this report to show where the Ada source code, software tools, intermediate data, models, and calculated values reside during the process. The application of the models is now performed at the MITRE-Washington Software Engineering Center using a Sun computer, running the Unix operating system. This directory structure can serve as a model that can be duplicated if another computer system is used to apply the models.

The report describes the series of steps invoking SAS programs that generate data files at the compilation unit, library unit, and subsystem levels. The library unit level files and subsystem level files will contain the quantities needed for calculating the values X_i so the models can be applied. The models described can be categorized based on: the quality factor (reliability, maintainability, or flexibility) associated with the metric predicted using the model; the level of granularity of the software quality predicted for subsystems or library unit aggregations; and the testing activities over which the model is predicting the metric either unit, system, and acceptance test or system and acceptance test . The report includes the steps to invoke SAS programs, corresponding to the models, to compute predicted values for quality factors.

ACKNOWLEDGMENTS

The authors acknowledge the contributions of Bradford T. Ulery to the development of the procedures and file directories described in this report.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

INTRODUCTION

This report describes procedures for applying Ada software quality prediction models for purposes of model validation. The multivariate regression models were developed under the Mission Oriented Investigation and Experimentation (MOIE) program of The MITRE Corporation. The models predict metrics related to software reliability, maintainability, and flexibility.

This section discusses the purpose of this report, background of the MOIE research that produced the models, the intended audience for this report, and an overview of the procedures to apply the models.

1.1 PURPOSE

This report is intended to support individuals who want to support in validating quality prediction models by applying them to Ada projects. The models have been developed in a research setting, based on software project data. The models are at a stage of development where they are ready for validation on additional software projects to refine the coefficients of the models. Validation of the models on diverse software projects will increase the confidence in subsequent application of the models.

The reader should be cautioned that the quality prediction models were developed using a particular set of Ada projects (described in [1]). The models should not be expected to be universally applicable regardless of the size and nature of the project. Indeed, understanding the range of applicability of the models is part of the validation process, which the procedures in this report are intended to facilitate.

1.2 BACKGROUND

MITRE has been conducting a MOIE research project investigating software quality prediction from Ada designs. The research has focused on prediction of metrics related to the software quality factors of reliability, maintainability, and flexibility. The project team has developed multivariate models that use characteristics of the Ada design as the basis for predictions of quality. The approach and rationale for developing the models are described in separate reports and technical papers. [1, 2, 4, 5]

The development of the quality prediction models involved the analysis of data from software development projects. The data included Ada source code and information on the experiences implementing and testing the code to make the software pass acceptance testing.

This information included reports of defects found during testing and reports of the effort expended to repair defects and to make changes to the software. Details of the Ada software and corresponding project data are discussed in [1].

For the models to be used with confidence, they need to be validated. For validation, the models need to be applied to projects other than those which were the basis of model development. This report describes the procedures for applying the models, so individuals outside the MOIE research team can participate in validating the models.

1.3 INTENDED AUDIENCE

This report is written for individuals who want to validate the quality prediction models by applying them to Ada source code. Another possible user of these procedures is someone who wants to conduct a static analysis of Ada source code. For someone interested only in static analysis, sections 1 through 4 will be sufficient to explain how to use the Ada Source Analyzer Program (ASAP) to produce a Project Summary Report (PSR), a sample of which is included in the Appendix to this report. Reference 3 provides detailed description of the capabilities of ASAP.

The procedures in this report assume a basic knowledge of Ada and familiarity with Unix commands.

1.4 OVERVIEW OF THE PROCEDURES

The starting point for someone to use this report is the availability of Ada source code and an interest in obtaining a static analysis of the code or applying the quality prediction models. The models are of the form

$$q = f(a_i * X_i),$$

where q is a quality factor to be predicted; a_i are coefficients resulting from the MOIE research and given in this report; and X_i are calculated quantities whose values depend on the Ada source code for the project. The values of X_i need to determined, so they can be combined with the coefficients a_i to produce the predicted quality factor. Validation involves comparing the predicted values to actual data as they become available on projects.

Sections 2 through 7 of this report describe the processing needed so that the values X_i can be calculated for given Ada source code. Section 8 gives the models themselves; that is, the coefficients a_i, and ways of combining ai and X_i to calculate the predicted quality factors. Section 8 also includes the commands to invoke programs that calculate the predicted values. Because the bulk of this report involves proceeding from Ada source code to the calculated quantities X_i, this process will be outlined in this section as an overview to Sections 2 through 7.

The models are based on static features of the Ada code, such as counts of declarations. Section 2 discusses the kinds of data on which the models depend. The discussion in Section 2 should help the reader understand what data is being extracted from the Ada code to use in later calculation of X_i. To extract these data, a software tool, ASAP, is used. ASAP is a static Ada source code analysis program developed at the University of Maryland. ASAP performs functions such as the following: presents profiles of compilation units; counts source lines and Ada statements; computes metrics based on Halstead software science analysis and McCabe cyclomatic complexity analysis; and prepares reports based on these analyses [3]. ASAP was developed as a stand-alone analysis tool. Not all of the extracted and calculated quantities produced by ASAP are needed for applying the quality prediction models; for example, Halstead and McCabe metrics are not used. ASAP was found to extract static data needed for our models so it is being used for that purpose in these procedures. Section 2 also defines the quality metrics that are based on the extracted data. These metrics relate to design characteristics and are elements of the models presented in Section 8.

Because several steps are involved in eventually calculating the X_i values, an organization of directories and files has been established and described in Section 3 to show where the Ada source code, software tools, intermediate data, models, and calculated values reside during the process. The application of the models is now performed at the MITRE-Washington (SWEC) using a Sun computer, running the Unix operating system. Section 3 discusses the directory organization on the SWEC computer. This directory structure can serve as a model that can be duplicated if another computer system is used to apply the models.

The steps involved in processing Ada source code, leading to the execution of the models, are depicted in Figure 1-1. The relationship of steps in the process to sections in this report is also shown in Figure 1-1. Section 4 describes the execution of ASAP to extract static data from the Ada source code. An additional product of this step is the production of the ASAP Project Summary Report, which provides a profile of the source code. An example of the ASAP Project Summary Report is included in the Appendix. For readers interested only in static analysis of their code, Section 4 contains the necessary commands leading to the generation of the Project Summary Report. For readers who plan to apply the quality prediction models, Section 4 also includes steps to execute additional extraction programs which operate on the output of ASAP to produce data files in an appropriate form for use with the (SAS), the statistical software used in the analysis.

As Figure 1-1 shows, Section 5 begins with all of the needed data available in SAS input files. The quantities needed for the quality prediction models refer to three different levels of structural granularity in the software. ASAP provides static data on Ada compilation units (as shown in Appendix A), so the first level of analysis is to calculate compilation-unit level measures. Section 5 discusses the steps involved in this processing, invoking SAS programs to generate the compilation unit files, as shown in Figure 1-1.

Compilation-unit level measures provide the needed data for measures at two higher levels of granularity: library unit aggregations (LUAs) and subsystems. SAS programs at the LUA and subsystem levels are discussed in sections 6 and 7, respectively. Quality prediction models have been established at these two levels, so the products of the processing in sections 6 and 7 directly feed the quality prediction models.

Unlike "compilation unit", the terms "library unit aggregation" and "subsystem" are not defined in the Ada language. Both terms arose from the MOIE research because of a need to express structural relationships at intermediate points between compilation units and entire systems, which may be extremely large. Both LUA and subsystem have been useful for analysis and reporting purposes. An LUA is an Ada library unit (LU) and its descendent compilation units, if any [2]. An LUA has become a structure of considerable interest in the research. The most interesting class of LUA examples consists of a package specification, a package body, and subunits. Such LUAs may include subunit structures which are nested at several levels.

Subsystem is used to retain a degree of generality in the research, when referring to major functional areas or principal units of a complete system. If the system is developed under Department of Defense (DOD)-STD-2167A, a subsystem may be a computer software configuration item, or computer software component. But, because such terms are not universally used, subsystem is used in this research.

At the conclusion of the steps described in sections 6 and 7, the library unit level files and subsystem level files will contain the quantities needed for calculating the values X_i. Section 8 represents the final stage of processing. The models described in Section 8 can be categorized based on: the quality factor (reliability, maintainability, or flexibility) associated with the metric predicted using the model; the level of granularity of the software quality predicted for subsystems or library unit aggregations; and the testing activities over which the model is predicting the metric -- either unit, system, and acceptance test (USA) or system and acceptance test (SA). For a detailed discussion of the background, motivation, rationale for the models, refer to references 1, 2, and 4. Section 8 includes the steps to invoke SAS programs, corresponding to the models, to compute predicted values for quality factors.

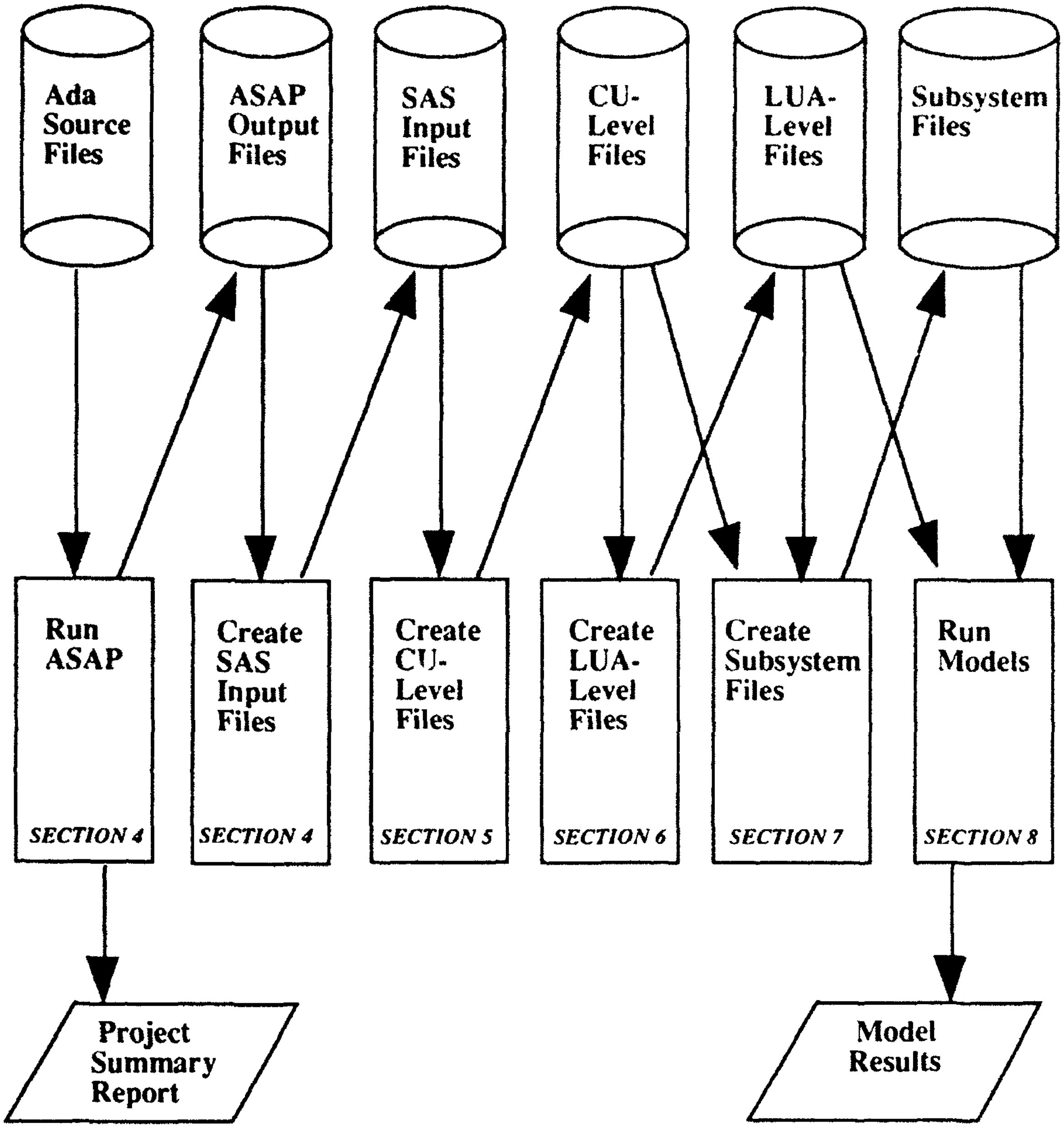

Figure 1-1. Overview of Procedures and Report Sections

SECTION 2

EXTRACTED DATA AND STRUCTURAL METRICS

The purpose of this section is to describe the classes of data that will be extracted from the Ada source code and the structural metrics used in the models. The procedures in sections 4 through 7 will refer to the data classes discussed in this section. The quality prediction models in Section 8 include factors based on the structural metrics.

The MOIE research has shown that the number and kinds of declarations are significant factors in quality prediction. Also significant are the patterns of sharing information by making quantities declared in one place accessible elsewhere in the software. We refer to declarations in the visible part of a library unit as being exported to a compilation unit that imports them by using a context ("with") clause. Data on declarations, exports, imports, and statement counts are extracted from the Ada source code and combined to form structural metrics that are used in the quality prediction models. This section defines the declarations, exports, imports, statement counts, and resulting structural metrics.

2.1 DECLARATIONS

Extracted data from the Ada source code includes counts of declarations by the following semantic classes: constants, objects, types, subtypes, formal parameters, exceptions, subprograms, packages, and tasks. Also figuring in the quality prediction models are counts of the total number of declarations, the number of program unit declarations, and the number of non-program units declarations.

The quality prediction models have also included factors that are sensitive to the possible use of the same identifier name in more than one way in the software. Data is extracted on the number of unique names declared, across the entire software and the number of unique names within a semantic class.

The example below illustrates the possible differences in these counts of declarations when names are used more than once.

```
Package P is
        procedure Q(A,B : in out integer; F: float);
        function F(X: integer) return integer;
        function F(X: character) return character;
        type T is new integer range 1..100;

end P;
```

The number of unique names declared does not include multiple declarations of the same name. In the example, F is declared twice as a function and once as a formal parameter, representing only one unique name declared (F). In the example, there are seven names declared (P, Q, A, B, F, X, T).

The number of unique names is also determined for each semantic class and then summed over all semantic classes. We call this count the number of unique declarations. In the example, F is declared twice as a function and once as a formal parameter, resulting in two unique declarations (i.e., a function F and a formal parameter F). The example has eight unique declarations: P, Q, A, B, F (function), F (formal parameter), X, T. Note that X is counted only once as a formal parameter.

Our third count of declarations is the total number of declarations including all overloaded names. In the example, F is declared twice as a function and once as a formal parameter, resulting in three total declarations (i.e., two functions named F and one formal parameter F). The example has ten total declarations (P, Q, A, B, F (integer function), F (character function), F (formal parameter), X (integer formal parameter), X (character formal parameter), T).

When declarations are divided into program unit and non-program unit declarations, the counts are not sensitive to the overloading. In the example, there are four program unit declarations (P, Q, F (integer function), F (character function)) and six non-program unit declarations made (A, B, F (formal parameter), X (integer formal parameter), X (character formal parameter), T).

2.2 EXPORTS

Exports are declarations made in the visible part of a library unit. Counts of exports are used in factors in various quality prediction models. Two counting rules for exports should be noted: (1) the name of a function or procedure implemented as a library unit is counted as a single declaration, since the declarations within the function or procedure are not visible, and (2) formal parameters to these subprograms, although visible, are not counted.

Figure 2-1 shows sample Ada source code consisting of library units P, Q, R, and S, and their associated secondary units. In this example, P exports five total declarations (P, T1, T2 ,T3,c), Q exports four declarations (Q, Q1, x,y), R exports six declarations (R, R1, R2, x, y, z), and S exports one declaration (S).

```
Package P is
    type T1 ...;
    type T2 ...;
    type T3 ...;
    c : constant = 10;
    end;

with P;
Package Q is
    procedure Q1(x,y : in out integer);
    end;

Package body Q  is
    procedure Q1(x,y : integer) is

    end;
end;

with Q;
Package R is
    procedure R1(x,y : in out integer);
    procedure R2(z : in out integer);
end;

Package body R is
    procedure R1 is separate;
    procedure R2 is separate;
end;

with P;
separate (R)
procedure R1(x,y : in out integer) is
...
end;

with P;
separate (R)
procedure R2(z : in out integer) is
...
end;

with P,R;
procedure S(a,b,c: in out integer) is
...
end;
```

Figure 2-1. Example to Illustrate Exports and Imports

A second count, related to exports, is the count of users of the library units. We count this in two ways: the number of CUs that contain a "with" to the LU in question, and the number of LUAs that contain a "with" to the LU in question. In the above example, P is "withed" by four CUs (package specification Q, subunit R. R1, subunit R. R2, procedure S), and thus three LUAs (Q, R, S); Q is withed by one CU (package spec R) and one LUA (R); R is withed by one CU (procedure S) and one LUA (S), and S is not withed at all.

2.3 IMPORTS

We associate a count of imports with each compilation unit based on the number of declarations in the visible part of the library units that are "withed in" to the CU (i.e., named in the CU's context clause). For example, we see that the compilation unit S imports the visible declarations of P (P, T1, T2, T3, c) and R (R, R1, R2, x, y, z). The other imports are as follows: package specification P imports nothing; package specification Q imports the five declarations from P (P, T1, T2, T3, c); package body Q imports nothing; package specification R imports the four visible declarations of Q (Q, Q1, x, y); package body R imports nothing; and subunits R.R1 and R.R2 each imports the five declarations of P (P, T1, T2, T3, c).

These counts of imports to each CU are then aggregated to the LUA level for CUs comprising the LUA. Three import counts are defined: total imports, unique imports, and cascaded imports.

Total imports is simply the sum of the imports for all CUs in the LUA. Thus, P imports nothing; Q imports five declarations; R imports 14 declarations; and S imports 11 declarations.

Unique imports is a count that is sensitive to multiple CUs importing the services of the same library unit. An example of this can be seen in the library unit aggregation R, where R.R1 and R.R2 each import the services of P. Unique imports do not count this duplication. Thus, P has no unique imports; Q has five; R has nine; and S has 11.

The third count of imports is the "cascaded imports" introduced in the MOIE research [1]. A declaration imported to one compilation unit will "cascade" through (i.e., be visible to) all descendent units of that compilation unit. For example, the five declarations imported to the package specification Q are also visible to the package body Q; thus, the library unit aggregation Q contains ten "cascaded imports:" the five directly imported to the specification, and the five cascaded to the body. P and S have no subunits, so the count of cascaded imports is the same as the count of total imports, namely, zero for P and 11 for S. Four declarations are imported to the package specification R; these are cascaded to the package body R and the subunits R.R1 and R.R2. Both R.R1 and R.R2 directly import five declarations, thus R has 26 cascaded imports.

2.4 STATEMENT COUNTS

ASAP provides various counts of source lines of code, comment lines, blank lines, and counts of Ada executable and declarative statements. For the most part, these counts are not included in our analyses. However, as a proxy for measures of the extent and uniformity of control flow, we defined several measures based on the number of call statements (either subprogram "call" (i.e., invocation) or task entry call) in the compilation units.

2.5 QUALITY METRICS

Based on the extracted data on declarations, imports, and exports, we have defined various metrics that relate to design characteristics and quality factors studied in the MOIE research. These metrics are included in the quality prediction models in Section 8. The metrics are defined in Table 2-1 (for subsystem-level metrics) and Table 2-2 (for LUA-level metrics) and discussed in References 1, 2, and 4.

Table 2-1. Subsystem-level Quality Metrics

Design Characteristic	Quality Metric
Context Coupling	IMPEXP: Number of unique declarations imported divided by the number of unique declarations exported
	WITHPLU: Mean number of library units "withed" per library unit aggregation
	PUDPLU: Mean number of imported program unit declarations per library unit aggregation
Control Coupling	CALLPSUB: Mean number of subprogram invocation statements per subprogram in the subsystem
	CALLPEX: Mean number of subprogram invocation statements per executable unit in the subsystem
Visibility	CIMPIMP: Number of unique cascaded declarations imported divided by the number of unique declarations imported
	VISHPUD: Percentage of hidden program unit declarations (i.e., number of hidden program unit declarations divided by number of hidden and visible program unit declarations)
	VISXPUD: Mean number of exported program unit declarations per library unit aggregation
Locality	FINTPUD: Percentage of imported program unit declarations originating in the same subsystem as the importing unit
Generality	GENS: Percentage of generic and instantiated library units in the subsystem
Parameterization	PARVPUD: Mean number of parameters per visible program unit

Table 2-2. LUA-level Quality Metrics

Design Characteristic	Quality Metric
Context Coupling	WITHS: Number of library units "withed" per LUA
Functionality	VIS PROG UNITS: Number of visible program units within the LUA

SECTION 3

DIRECTORIES AND FILE ORGANIZATION

This section describes the directory structure that has been established to facilitate the application of the quality prediction models. This directory structure has been implemented on the SWEC Sun host computer named National under Unix. There are two purposes for describing the directories and files: (1) They are referenced in the processing steps as locations for intermediate data and results. Readers who are applying the quality prediction models in the MITRE SWEC will know where to look for those data or results; and, (2) The MOIE research team has found this directory structure to be a useful way to organize the potentially confusing collection of programs and data. If the quality prediction models are implemented on a different host computer, this file structure may be helpful as a model.

Figure 3-1 depicts the Unix directory structure. The "qmtop" directory is accessed through the "design1" directory. The "qm" of qmtop stands for quality metrics. The qmtop directory provides access to the projects, tools, and templates directories. Table 3-1 describes the directories shown in Figure 3-1. The projects directory provides access to directories for specific projects, while tools like ASAP and SAS are contained in the tools directory.

Many of the SAS programs needed for application of the quality prediction models create files associated with the project to which the models are being applied. To retain flexibility in these procedures, so they can be applied to various projects, templates have been written by the MOIE team. The templates are generic programs that, when supplied with a project name, can produce specific instantiated programs to create files associated with the project name.

Figures 3-2 through 3-4 and Tables 3-2 through 3-4 display and describe the directories for projects, tools, and templates.

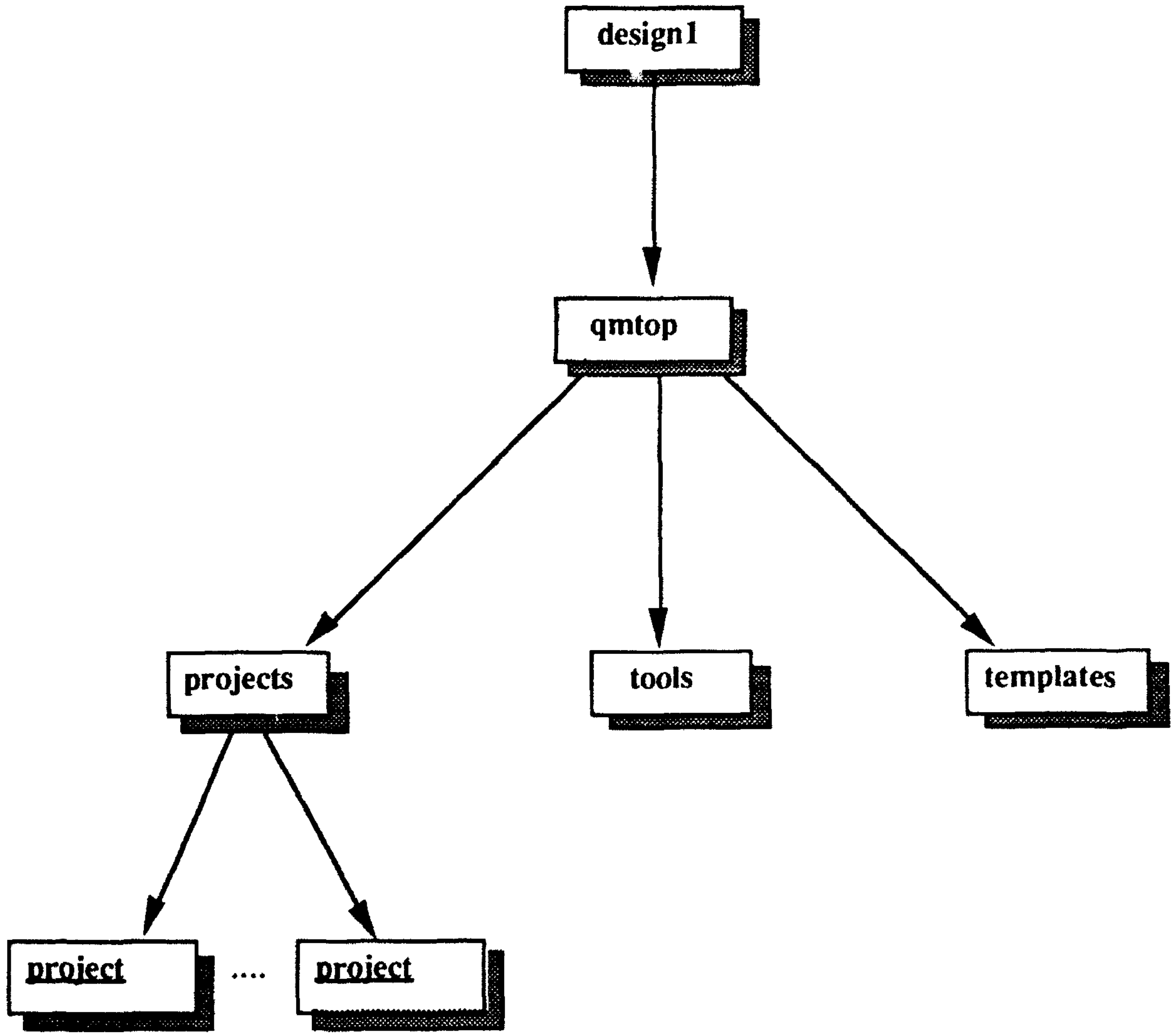

Figure 3-1. High Level Directories

Table 3-1. High Level Directories

Directory Name	Directory Description
design1	This directory is at the highest level on the Sun host called National and provides access to the qmtop directory.
qmtop	This directory is the highest directory for the quality metrics project. It incorporates directories that include programs and data input and data output for ASAP, SAS, and MITRE-developed quality prediction models.
projects	This directory contains directories for individual projects that are to be analyzed.
<u>project</u> *(Throughout the remainder of this report when the term project is underlined (<u>project</u>) it should be interpreted as an actual project name)*	This directory incorporates directories that include project-specific data and programs.
tools	This directory incorporates directories that include executable programs and source code. These programs are used to perform ASAP and SAS analyses, and to apply the quality prediction models.
templates	This directory incorporates directories that include generic SAS programs that can be used to create project-specific programs.

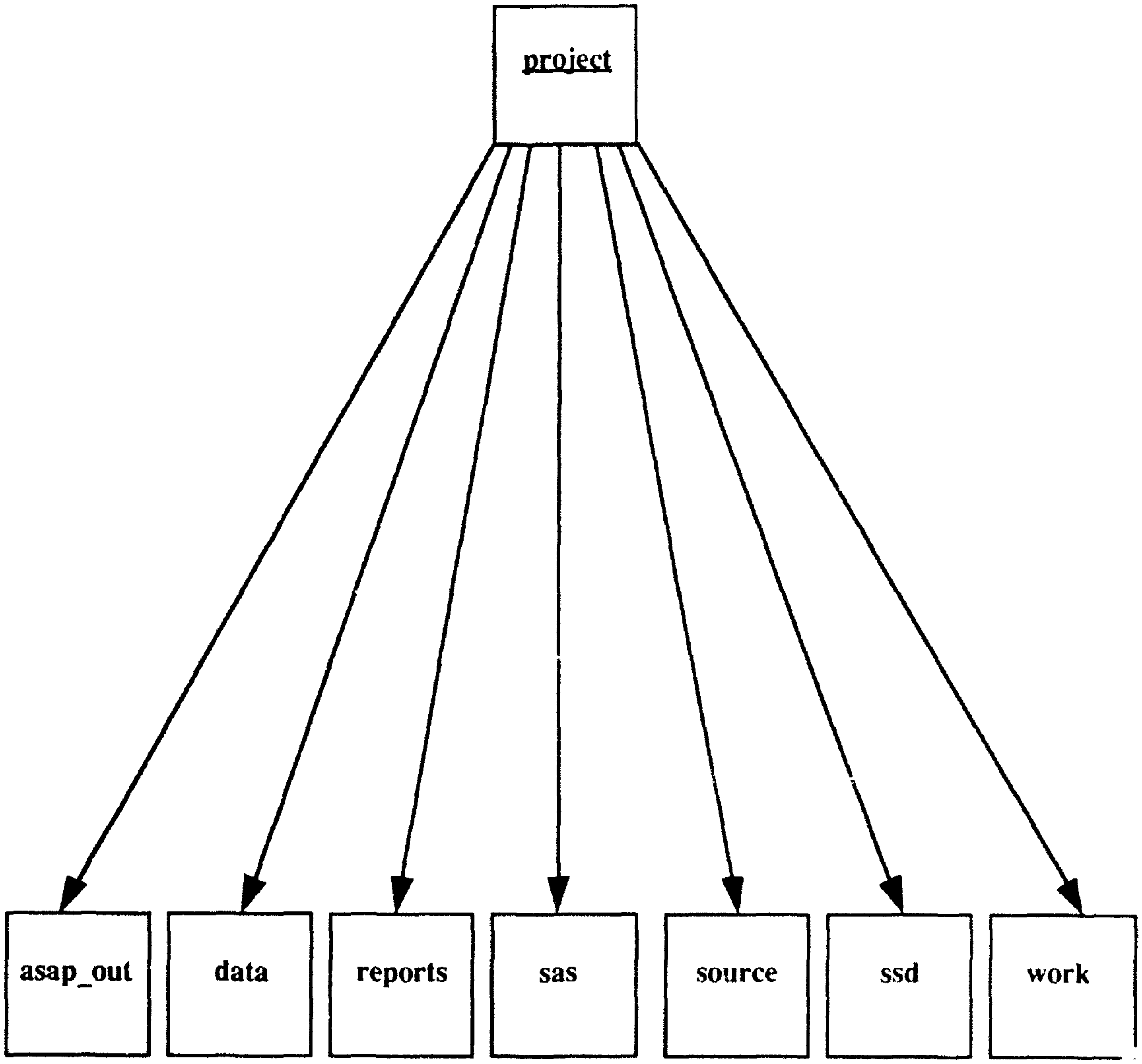

Figure 3-2. Project Directories

Table 3-2. Project Directories

Directory Name	Directory Description
asap_out	Output files created when running ASAP are placed in this directory. This directory contains one output file for each source input file to ASAP.
data	ASAP data output files are placed in this directory.
reports	Files that include predictions and summary descriptions are placed in this directory.
sas	SAS programs obtained from the templates directory and modified to be project-specific are placed in this directory.
source	Ada source code files are contained in this directory.
ssd	SAS data sets are placed in this directory when they are created by SAS programs.
work	General work files can be placed in this directory.

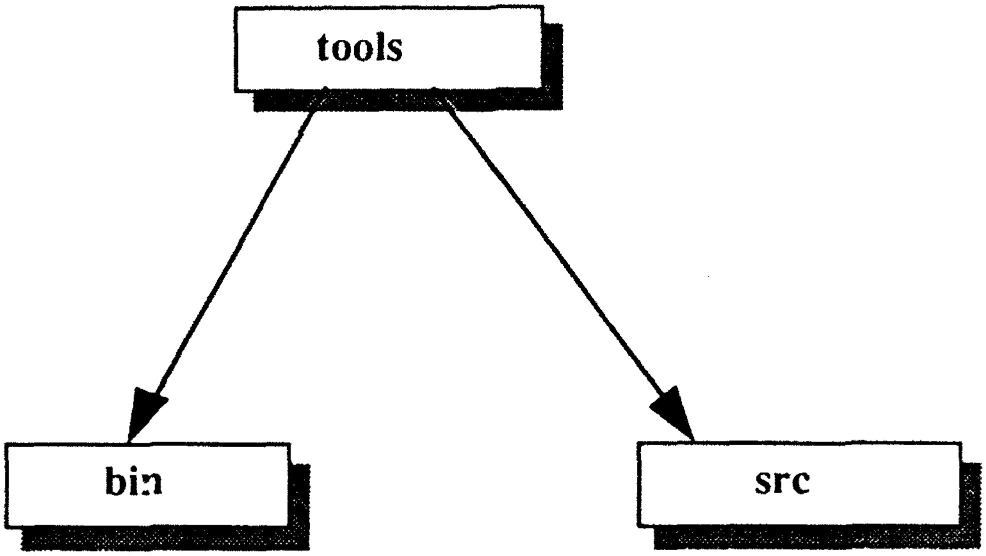

Figure 3-3. Tools Directories

Table 3-3. Tools Directories

Directory Name	Directory Description
bin	This directory contains executable programs.
src	This directory contains directories that contain source code associated with the executable programs included in the bin directory.

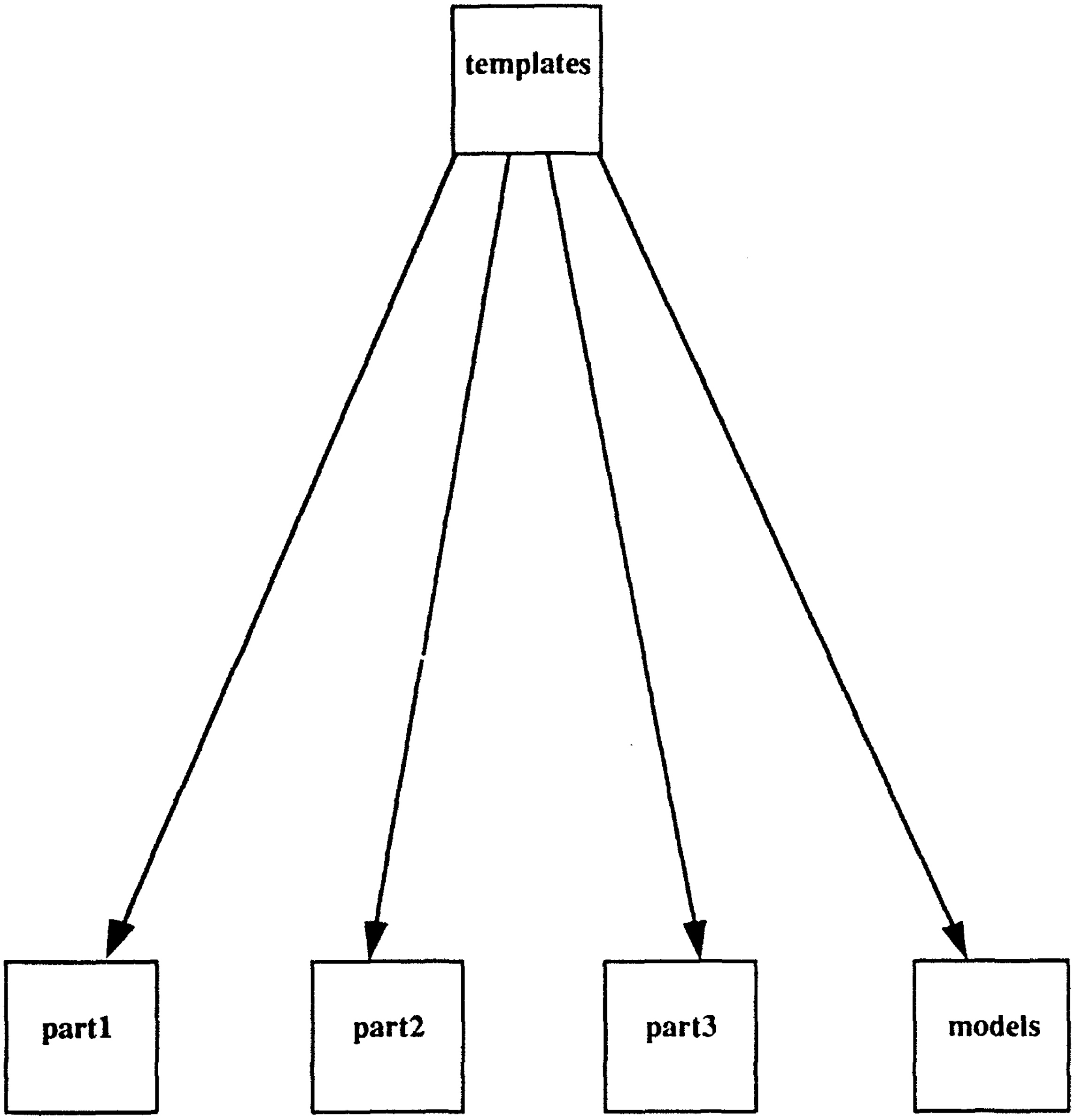

Figure 3-4. Template Directories

Table 3-4. Template Directories

Directory Name	Directory Description
part1	This directory contains SAS programs that are used to create the project's SAS database.
part2	This directory contains SAS programs that are used to create data sets at the library unit level .
part3	This directory contains SAS programs that are used to create data sets at the subsystem level.
models	This directory contains programs that are used to produce results from the quality prediction models.

SECTION 4

ADA SOURCE CODE ANALYSIS

This section describes the procedure for using ASAP and other programs to extract data from Ada source code and create files that are in the proper format for use with SAS. ASAP also can generate a Project Summary Report. This section contains an overview of the procedure and the detailed steps required.

4.1 OVERVIEW OF THE SOURCE CODE ANALYSIS PROCEDURE

The starting point for using the procedure in this section is the presence of the Ada source code for a project. The steps in the procedure are as follows:

- Step 1-1 Establish user path and directories
- Step 1-2 Create the ASAP database
- Step 1-3 Create the project summary file (and report)
- Step 1-4 Create "withs by CU" file
- Step 1-5 Create instantiations file
- Step 1-6 Create declarations file
- Step 1-7 Create filenames and CUs file
- Step 1-8 Create CU call counts file
- Step 1-9 Create filename/subsystem mapping file
- Step 1-10 Change file data to uppercase

Step 1-2 runs ASAP, creating output files used in subsequent steps. Step 1-3 uses the ASAP output files to create a project summary file that can be printed as a Project Summary Report (see the Appendix for an example). Steps 1-4 through 1-9 create files that are subsequently used to create SAS files. The final step changes data in the created files to uppercase so that the files are in the correct form for SAS processing.

Figure 4-1 highlights the directories involved in the procedure, as follows:

- bin: contains programs used in steps 1-2 through 1-10
- source: contains source code used as input to step 1-2
- asap_out: contains input data for steps 1-4, 1-5, 1-6, 1-7, 1-8, and 1-10; and files of output data from step 1-2
- data: contains input data for step 1-3; and files of output data from steps 1-2 through 1-10.

After following this procedure, all needed data will be present in SAS input files, ready for compilation unit level analysis in Section 5.

4.2 DETAILED SOURCE CODE ANALYSIS PROCEDURE

Step 1-1 Establish user path and directories

The procedures begin with steps to ensure that the Unix path and directories are established, so that commands in this section will execute correctly.

The user path should be updated to include the following directory:

/design1/qmtop/tools/bin

A new project directory must be created in the projects directory. As previously indicated (Table 3-1), the term project is used throughout this report to represent the name of a project. The following lower level directories must also be created in the newly created project directory: asap_out, data, reports, sas, source, ssd, and work. The Ada source code must be placed in the source directory located in the project directory. The remaining steps in this section assume the current directory is the project directory.

Step 1-2 Create the ASAP database

This step takes the Ada source code data located in the source directory and creates ASAP reports stored in the asap_out directory and also stored in the data directory. The ASAP database file reports are used in subsequent steps when extracting data to be used as input to SAS. The database file is used in the next step to create a file that can be used to produce the Project Summary Report.

The command to initiate this process follows:

datasap source asap_out data/project.db

This command consists of four parts: the program name (datasap); the input directory name (source); an output directory name (asap_out); and a second output directory name along with the name of the file to be stored in the directory (data/project.db).

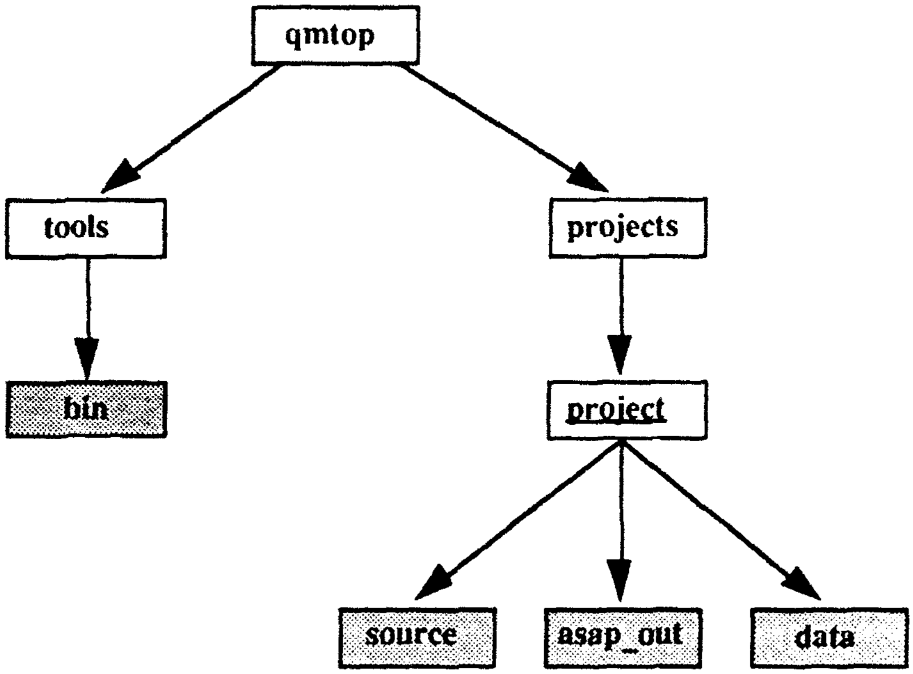

Figure 4-1. Directories Used in Source Code Analysis Procedures

Step 1-3 Create the project summary file and report

This step uses the previously generated ASAP database file to create a project summary file, which is formatted for printing as the Project Summary Report(project.usumm). A sample of this report is shown in the Appendix. A second file is also packed, project.summ, which is used in the SAS processing described in Section 5.

The command to initiate this process follows:

projsumm data/project.db data/project

This command consists of three parts: the program name (projsumm); the input directory and file name (data/project.db); and the output directory and file name (data/project).

Step 1-4 Create "withs by CU" file

This step takes files created in step 1-2 and creates an output file (project.withs) that is used during the SAS processing described in Section 5.

The command to initiate this process follows:

```
withextr asap_out > data/project.withs
```

This command consists of four parts: the program name (withextr); the input directory name (asap_out); the Unix symbol directing outputs to a file (>); and the name of the output directory along with the output file stored in the directory (data/project.withs).

Step 1-5 Create instantiations file

This step takes files created in step 1-2 and creates an output file (project.insts) that is used during the SAS processing described in Section 5.

The command to initiate this process follows:

```
instextr asap_out > data/project.insts
```

This command consists of four parts: the program name (instextr); the input directory name (asap_out); the Unix symbol directing outputs to a file (>); and the name of the output directory along with the output file stored in the directory (data/project.insts).

Step 1-6 Create declarations file

This step takes files created in step 1-2 and creates an output file (project.decs) that is used during the SAS processing described in Section 5.

The command to initiate this process follows:

```
decsextr asap_out > data/project.decs
```

This command consists of four parts: the program name (decsextr); the input directory name (asap_out); the Unix symbol directing outputs to a file (>); and the name of the output directory along with the output file stored in the directory (data/project.decs).

Step 1-7 Create filenames and CUs file

This step takes files created in step 1-2 and creates an output file (project.fnmap) that contains a mapping of filenames to compilation unit names. This output is used during the SAS processing described in Section 5.

The command to initiate this process follows:

```
fntocu asap_out > data/project.fnmap
```

This command consists of four parts: the program name (fntocu); the input directory name (asap_out); the Unix symbol directing outputs to a file (>); and the name of the output directory along with the output file stored in the directory (data/project.fnmap).

Step 1-8 Create CU call counts file

This step takes files created in step 1-2 and creates an output file (project.calls) that contains data concerning compilation unit counts. This output is used in during the SAS processing described in Section 5.

The command to initiate this process follows:

callextr asap_out > data/project.calls

This command consists of four parts: the program name (callextr); the input directory name (asap_out); the Unix symbol directing outputs to a file (>); and the name of the output directory along with the output file stored in the directory (data/project.calls).

Step 1-9 Create filename/subsystem mapping file

This step requires the user to create a file named project.ssmap that contains two columns. The first column containing the name of the Ada source file and the second the "subsystem" (as discussed in Section 1) to which the file belongs. After the file is created, it is placed in the data directory. In the event that subsystems have not been identified for the project, the user can map all files to a single subsystem. This will allow the user to continue with the processing described in the next section.

Step 1-10 Change file data to uppercase

This step takes the files created in steps 1-4 through 1-9 and changes data to uppercase so that the files can be used to create SAS files.

The command to initiate this process follows:

uppercase project

This command consists of two parts: the program name (uppercase); and the input directory name (project).

SECTION 5

COMPILATION UNIT LEVEL ANALYSIS

This section describes a procedure for creating project SAS files at the compilation unit level. The procedure involves executing SAS programs that operate on the SAS input files created by the procedure in Section 4. This section contains an overview of the procedure and the detailed steps in the processing.

5.1 OVERVIEW OF THE COMPILATION UNIT LEVEL ANALYSIS PROCEDURE

The starting point is the completion of the procedure in Section 4, resulting in SAS input files. The steps in the procedure are as follows:

- Step 2-1 Create SAS programs from templates
- Step 2-2 Remove duplicate CU names
- Step 2-3 Create CU file
- Step 2-4 Create instantiations file
- Step 2-5 Create declarations file
- Step 2-6 Create declaration counts file
- Step 2-7 Create "withed in" relationship file
- Step 2-8 Create the SAS database file

When running SAS from the UNIX command line (as we describe in this report), for each SAS program run (e.g., sas *sas_program)* an output file *sas_program*.log will be generated. This file contains a log of the executed SAS program, including any warnings and error messages. It is recommended that this file be examined after each step to ensure that no errors have been encountered.

A second output is often produced, *sas_program*.lst. This file contains the output from any SAS print procedures. Many of the steps produce intermediate reports that are contained in these files. While examining these reports is not necessary to obtain the predictive results, it can help to provide a better understanding of the system under analysis and help to resolve any errors that may have been encountered.

The first step uses program templates to create project-specific programs that will be used during subsequent steps in this process. The next step examines previously generated files to determine whether duplicate compilation unit names exist. If duplicate names exist it is necessary to take steps to eliminate the duplicates. The remaining steps take either previously generated files or files created during this process to create SAS CU-level files.

Figure 5-1 identifies the directories that are used during this procedure, as follows:

- templates: contains programs used as input to step 2-1
- sas: contains programs that are the output of step 2-1 and used in steps 2-2 through 2-8
- data: contains data used as input to steps 2-2 through 2-5, 2-7, and 2-8
- ssd: contains data used as input to step 2-6 and data used as output from steps 2-3 through 2-8
- bin: contains the program used in step 2-1

After following this procedure, all needed data is in CU-level files, to be used for library unit level analysis (Section 6) and subsystem level analysis (Section 7).

5.2 DETAILED COMPILATION UNIT LEVEL ANALYSIS PROCEDURE

Step 2-1 Create SAS programs from templates

Prior to beginning the steps described in this section, it is assumed that the procedure described in Section 4 has been completed. Thus the data directory contains the SAS input files (e.g., project.decs, project.withs, and project.insts) that will be used during this procedure. It is further assumed that the current directory is any of the following: design1, qmtop, tools, or bin.

This step instantiates program templates that are stored in the templates directory to create programs specific to the project of interest. The resulting programs are stored in the sas directory. The templates must be instantiated so that the programs will be able to access files that include the project identification as part of the name (e.g., project.withs) and to store results in the proper location.

The command to initiate this process follows:

```
modify_templates project
```

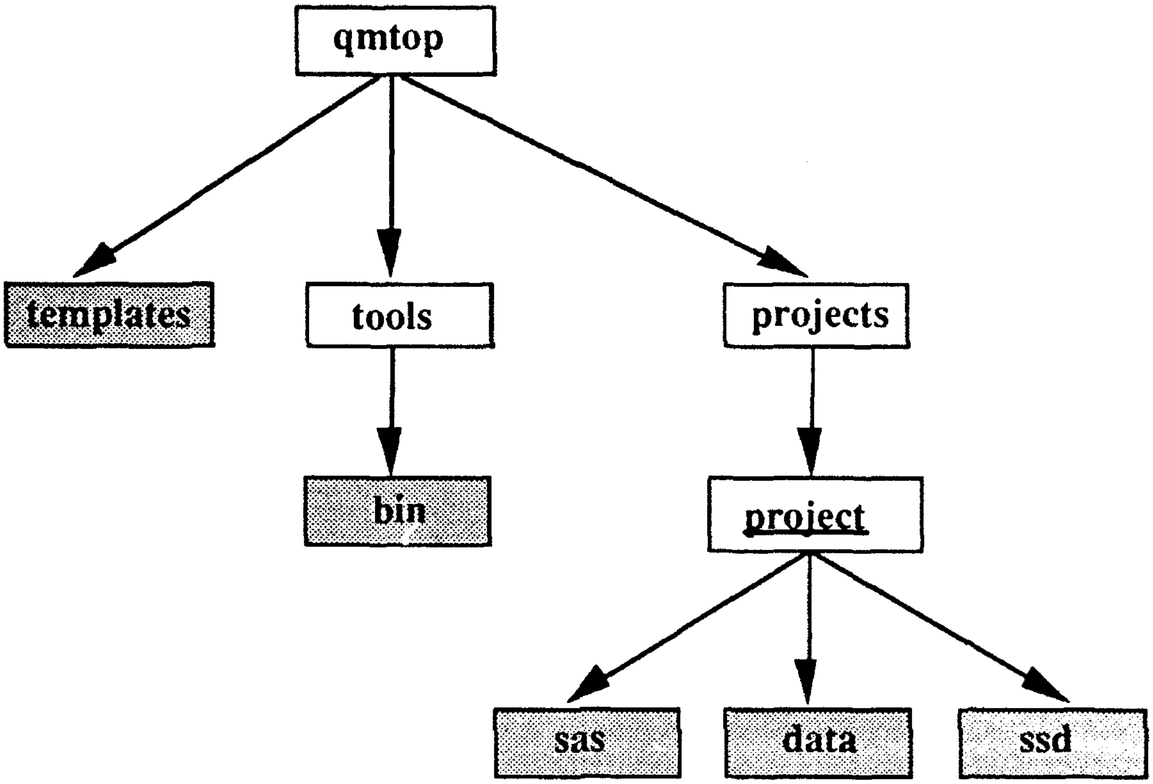

Figure 5-1. Directories Used in CU-level Analysis Procedure

This command contains the name of a program (modify_templates) and a parameter which is the name of the project that is being analyzed. The input template programs for this process are located in the part1, part2, part3, and models directories located in the qmtop/templates directory. The outputs from this process are stored in the part1, part2, part3, and models directories located in the sas directory.

Step 2-2 Remove duplicate CU names

For this step and remaining steps in this procedure, the current directory should be the part1 directory in the sas directory.

This step examines the previously generated project.withs file for duplicate compilation unit names. It produces an output file that the user must examine to determine if any compilation unit names occur more than once. If duplicate compilation units are discovered, the user must go back to the original code and eliminate the duplicate code and then restart the process with the steps described in Section 4.

The command to initiate this process follows:

sas dupdecs1.sas

This command invokes a program that checks for duplicate compilation unit names for any project. The project.withs file is an input to this program. If duplicate compilation unit names exist, they are output to the sas_dupdecs1.lst file.

A second approach to detecting duplicates is to examine the previously generated project.decs file for duplicate compilation unit names. If duplicate compilation units are discovered, the user must go back to the original code and eliminate the duplicate code and then restart the process with the steps described in Section 4.

The command to initiate this process follows:

sas dupdecs2.sas

This command invokes a program that also checks for duplicate compilation unit names for any project. The project.decs file is an input to this program. If duplicate compilation unit names exist they are output to the sas_dupdecs2.lst file.

Step 2-3 Create CU file

This step takes the previously generated project.withs file and creates a SAS file (culist.ssd01) that contains the names of compilation units and their types.

The command to initiate this process follows:

sas culist.sas

This command invokes a program that takes the project.withs file containing relationships between user compilation units and "withed in" library units and generates the culist.ssd01 file that contains the names of the compilation units and their types. The output file is contained in the ssd directory.

Step 2-4 Create instantiations file

This step takes the previously generated project.insts file and creates a SAS file (instssf.ssd01) that contains the instantiations associated with each compilation unit. The output from this step serves as an input to step 2-6.

The command to initiate this process follows:

sas instssf.sas

This command invokes a program that creates the instssf.ssd01 file. The input file to this program is the project.insts file.

Step 2-5 Create declarations file

This step takes the previously generated project.decs file and creates a SAS file (decssf.ssd01) that contains information concerning declarations. The output from this step serves as an input to the following step.

The command to initiate this process follows:

sas decssf.sas

This command invokes a program that creates the decssf.ssd01 file containing declarations information.

Step 2-6 Create declaration counts file

This step provides the additional processing needed to calculate counts that are sensitive to names being used more than once, as discussed in Section 2.1. This step takes the file generated by the previous step (decssf.ssd01) and creates a SAS file (psdecnum.ssd01) that contains the number of declaration names, the number of overloaded declaration names, and the number of unique declaration name/class combinations.

The command to initiate this process follows:

sas psdecnum.sas

This command sends its results to the file, psdecnum.ssd01. That file and the file produced by step 2-4 (instssf.ssd01) are then input to another program, invoked by the command, psdecmod.sas, yielding the required declaration counts data in the psdecmod.ssd01 file.

Step 2-7 Create "withed in" relationship file

This step takes the previously generated project.withs file containing relationships between user compilation units and "withed in" library units and creates a SAS file (wither.ssd01).

The command to initiate this process follows:

sas with1.sas

This command invokes a program that takes the project.withs file and creates the wither.ssd01 file.

Step 2-8 Create the SAS database file

This step takes the previously generated project.summ file and creates the SAS database file (database.ssd01).

The command to initiate this process follows:

sas database.sas

This command contains the name of a program that takes the project.summ file and generates the database.ssd01 file. The procedure is now complete with the output file containing information on counts of declarations and "with's" at the compilation unit level.

SECTION 6

LIBRARY UNIT LEVEL ANALYSIS

This section describes a procedure for creating project SAS files at the library unit aggregation (LUA) level. The procedure involves executing SAS programs that operate on the SAS compilation unit level data created by the procedure in Section 5. This section contains an overview of the procedure and the detailed steps in the processing.

6.1 OVERVIEW OF THE LIBRARY UNIT LEVEL ANALYSIS PROCEDURE

The starting point is the completion of the procedure in Section 5, resulting in SAS compilation unit level data. To create library unit level files, the steps in the procedure are as follows:

- Step 3-1 Create compilation unit mapping file
- Step 3-2 Create counts of "withed in" declarations file
- Step 3-3 Create imports file
- Step 3-4 Create unique imports counts file
- Step 3-5 Create the exports file
- Step 3-6 Create cascaded imports (CU level) file
- Step 3-7 Create cascaded imports (LU level) file
- Step 3-8 Create "withed by" file
- Step 3-9 Create program unit declarations file
- Step 3-10 Combine previously generated files
- Step 3-11 Create library unit files

This section describes a procedure to create library unit level SAS files. The first step creates a file that maps compilation units to subsystem units. This step uses files created during the processing described in Section 4. Steps 3-2 through 3-9 use files created during processing described in Section 5 to create library unit files for various categories of data (e.g., "withed in" data, cascaded imports data, and unique imports data). Step 3-10 combines the files generated in steps 3-2 through 3-9. Step 3-11 takes the file generated in step 3-10 and creates SAS files at the library unit level.

Figure 6-1 identifies the directories that are used during this procedure, as follows:

- sas: contains the programs used in all steps
- data: contains data used as input to step 3-1
- ssd: contains data used as input to steps 3-2 through 3-1; and files used as output data from step 3-11

After following this procedure, the library unit level files are complete.

6.2 DETAILED LIBRARY UNIT LEVEL ANALYSIS PROCEDURE

This section describes each step in the procedure by giving the commands used to invoke the necessary programs and the input and output.

Step 3-1: Create Compilation Unit Mapping File

The starting point for using the procedure in this section is the existence of the following specific project files:

- in the data directory: project.fnmap (from step 1-7) and project.ssmap (from step 1-9)
- in the ssd directory: wither.ssd01 (from step 2-7) and psdecmod.ssd01 (from step 2-6).

For this step and remaining steps in this procedure, the current directory should be the part2 directory in the sas directory.

This step takes a previously generated file that contains a mapping of filenames to compilation unit names and another previously generated file that contains the mapping of filenames to subsystems and creates a file that maps compilation units to subsystem units.

The command to initiate this process follows:

```
sas cumap.sas
```

This command contains the name of a program that takes the project.fnmap and project.ssmap files and creates the cumap.ssd01 file. The output contains a mapping of compilation units to subsystem units.

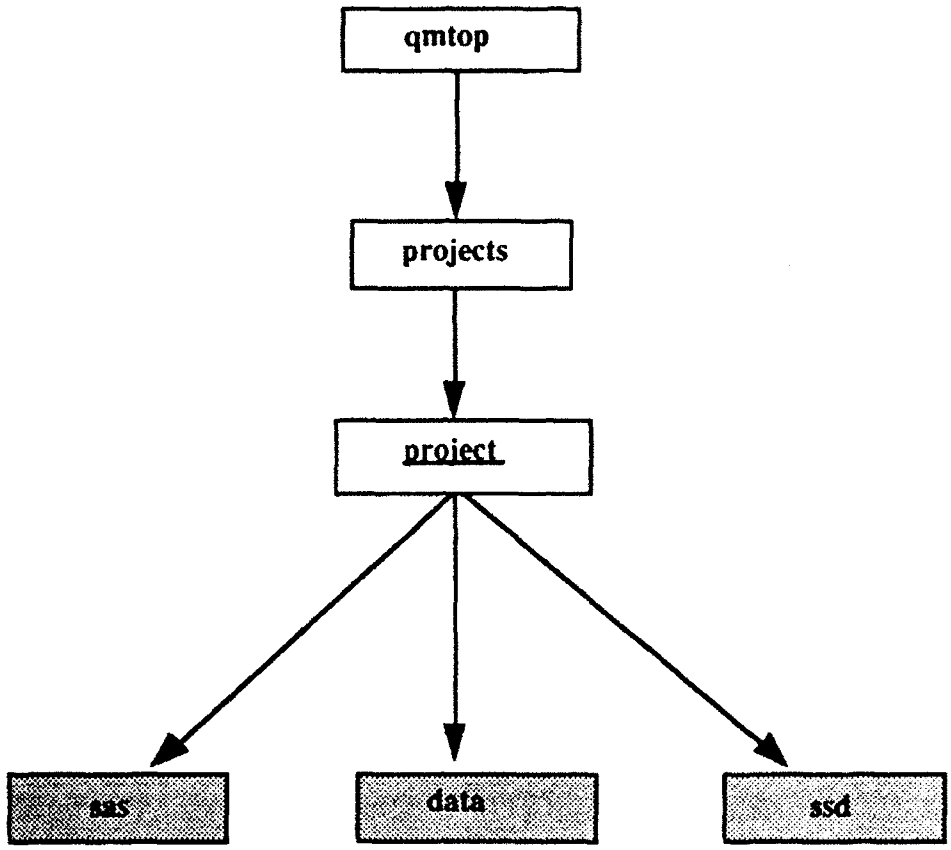

Figure 6-1. Directories Used in LU-level Analysis Procedure

Step 3-2: Create Counts of "Withed In" Declarations File

This step takes previously generated files and creates a file that contains counts of "withed in" declarations for library units. The output file is used as an input to step 3-10.

The command to initiate this process follows:

sas lusswdec.sas

This command contains the name of a program that outputs counts of the declarations withed into the various compilation units and is ordered by subsystem. Note that the declarations from the standard library are not included in these counts. The input files are wither.ssd01, cumap.ssd01, and psdecmod.ssd01. The output file is withdecs.ssd01.

Step 3-3: Create Imports File

This step takes the file created in step 3-2 and creates a file that contains the counts of imports from external subsystems (i.e., subsystems other than the one containing the library unit in question) at the library unit level. Note that, in this step, if a library unit is "withed in" more than once into compilation units of a LUA, the imports will be counted more than once. The output file is used as an input to step 3-10.

The command to initiate this process follows:

sas luimp.sas

This command contains the name of a program that uses the withdecs.ssd01 file to calculate the number of imports from external subsystems at the library unit level. The output files from this program are lusimp.ssd01 and lunimp.ssd01.

Step 3-4: Create Unique Imports Counts File

This step takes the file created in step 3-2 and creates a file that contains the counts of imports from external subsystems at the library unit level. Note that, in this step, if a library unit is "withed in" more than once into compilation units of a LUA, the imports will be counted only once. The output files are used as inputs to step 3-10.

The command to initiate this process follows:

sas luuimp.sas

This command contains the name of a program that uses the withdecs.ssd01 file to calculate imports from external subsystems at the library unit. The output files from this program are luusimp.ssd01 (imports from the same subsystem) and luunimp.ssd01 (imports from external subsystems).

Step 3-5: Create the Exports File

This step takes the file created in step 3-2 and creates a file that contains counts of exports at the LUA level. The output file is used as an input to step 3-10.

The command to initiate this process follows:

sas luexp.sas

This command contains the name of a program that computes the exports at the LUA level. The input file to this program is withdecs.ssd01. The output file from this program is luexps.ssd01.

Step 3-6: Create Cascaded Imports (Compilation Unit Level) File

This step takes the files created in steps 3-1 and 3-2 and creates a file that contains cascaded import counts for compilation units. The output file is used as an input to step 3-7.

The command to initiate this process follows:

sas lucscd.sas

This command contains the name of a program that uses the withdecs.ssd01 and cumap.ssd01 files to compute cascaded imports. The output file is lucscd.ssd01

Step 3-7: Create Cascaded Inputs (Library Unit Level) File

This step takes the file created in step 3-3 and creates a file that contains counts of cascaded imports at the LUA level. The output file is used as an input to step 3-10.

The command to initiate this process follows:

sas lucimp.sas

This command contains the name of a program that uses the lucscd.ssd01 file to compute the cascaded imports at the LUA level. The output file from this program is lucimp.ssd01.

Step 3-8: Create "Withed By" File

This step takes the previously generated file containing "withed in" information and the previously generated compilation unit mapping file and creates a "withed by" file for library units.

The command to initiate this process follows:

sas luwithby.sas

This command contains the name of a program that calculates the "withed by" relationships. The input files to this program are wither.ssd01 and cumap.ssd01. The output file from this program is luwithby.ssd01.

Step 3-9: Create Program Unit Declarations File

This step takes the previously generated database file and the previously generated compilation unit mapping file and creates a file that contains information on parent-child relationships (e.g., package body-subunit [1]) and on the counts of various program unit declarations within a LUA. The output file is used as an input to step 3-10.

The command to initiate this process follows:

sas lupuds.sas

This command contains the name of a program that takes as input the SAS data set contained in database.ssd01 and cumap.ssd01 and delivers as output the lupuds.ssd01 file.

Step 3-10: Combine Previously Generated Files

This step takes files created by steps 3-2 through 3-9 and combines this data into a single data file. The output from this step serves as input to step 3-11.

The command to initiate this process follows:

sas lusscmb.sas

This command contains the name of a program that creates a SAS data set for the specified project at the library unit level. It combines the following files:lunimp.ssd01, lucimp.ssd01, lusimp.ssd01, luexp.ssd01, luunimp.ssd01, luusimp.ssd01, lupuds.ssd01, luwithby.ssd01.

The output file is lucmb.ssd01.

Step 3-11: Create Library Unit Data Sets

This step takes the file generated by the previous step and generates a file, ludb.ssd01, containing data on all LUAs and also creates four files that represent partitions of ludb.ssd01 according to characteristics of the LUAs. The first of these files, pkgs.ssd01, contains information concerning LUAs that contain a library unit package and at least one executable line of code. A second file, pkgd.ssd01, contains information concerning LUAs that contain a library unit package, but do not contain any executable lines of code. The third file, subs.ssd01, contains information concerning LUAs that are subprograms. The fourth file, inst.ssd01, contains information concerning LUAs that are instantiations of generics.

The command to initiate this process follows:

sas ludb.sas

This command contains the name of a program that creates SAS data sets at the LUA level. The input file to the program is lucmb.ssd01. The output files from this program are ludb.ssd01, pkgs.ssd01, pkgd.ssd01, subs.ssd01, and inst.ssd01.

SECTION 7

SUBSYSTEM LEVEL ANALYSIS

This section describes a procedure for creating project SAS files at the subsystem level. The procedure involves executing SAS programs that operate on the SAS compilation unit level and library unit level data created by the procedures in section 5 and 6. This section contains an overview of the procedure and the detailed steps in the processing.

7.1 OVERVIEW OF THE SUBSYSTEM LEVEL ANALYSIS PROCEDURE

The starting point is the completion of the procedure in Section 6, resulting in SAS library unit level data. To create subsystem level files, the steps in the procedure are as follows:

- Step 4-1 Create subsystem exports file
- Step 4-2 Create subsystem imports file
- Step 4-3 Create subsystem program unit declarations file
- Step 4-4 Create subsystem level SAS file

This section describes a procedure to create subsystem level file. The first three steps take previously generated files and create files containing data concerning exports, imports, and program unit declarations at the subsystem level. The last step takes the files created by the previous steps and combines the data into a single file.

Figure 7-1 identifies the directories that are used during this procedure, as follows:

- sas: contains the programs used in all steps
- data: contains data used as input and output for all steps

After following this procedure, the subsystem level files are complete.

7.2 DETAILED SUBSYSTEM LEVEL ANALYSIS PROCEDURE

This section describes each step in the procedure by giving the commands used to invoke the necessary programs and the input and output.

The starting point for this procedure is the existence of the files withdecs.ssd01 and database.ssd01.

Step 4-1 Create subsystem exports file

For this step and remaining steps in this procedure, the current directory should be the part3 directory in the sas directory.

This step takes the previously generated file containing information concerning "withed in" declarations at the LUA level and creates several files containing information concerning exports at the subsystem level. The outputs from this step serve as input to step 4-4.

The command to initiate this process follows:

sas ssexpts.sas

This command contains the name of a program that creates SAS data sets at the subsystem level. The input file to the program is withdecs.ssd01. The output files from this program are exports.ssd01, exptsubs.ssd01, ngenexsb.ssd01, expdecs.ssd01, and crssdecs.ssd01.

Step 4-2 Create subsystem imports file

This step takes the previously generated file containing information concerning "withed in" declarations at the LUA level and creates several files containing information concerning imports at the subsystem level. The outputs from this step serve as input to step 4-4.

The command to initiate this process follows:

sas ssimpts.sas

This command contains the name of a program that creates SAS data sets at the subsystem level. The input file to the program is withdecs.ssd01. The output files from this program are impcus.ssd01, impdecs.ssd01, impcscd.ssd01, and impcdec.ssd01.

Step 4-3 Create subsystem program unit declarations file

This step takes the previously generated database file and creates two files that contain data concerning program unit declarations at the subsystem level. The outputs from this step serve as inputs to step 4-4.

The command to initiate this process follows:

sas sspuds.sas

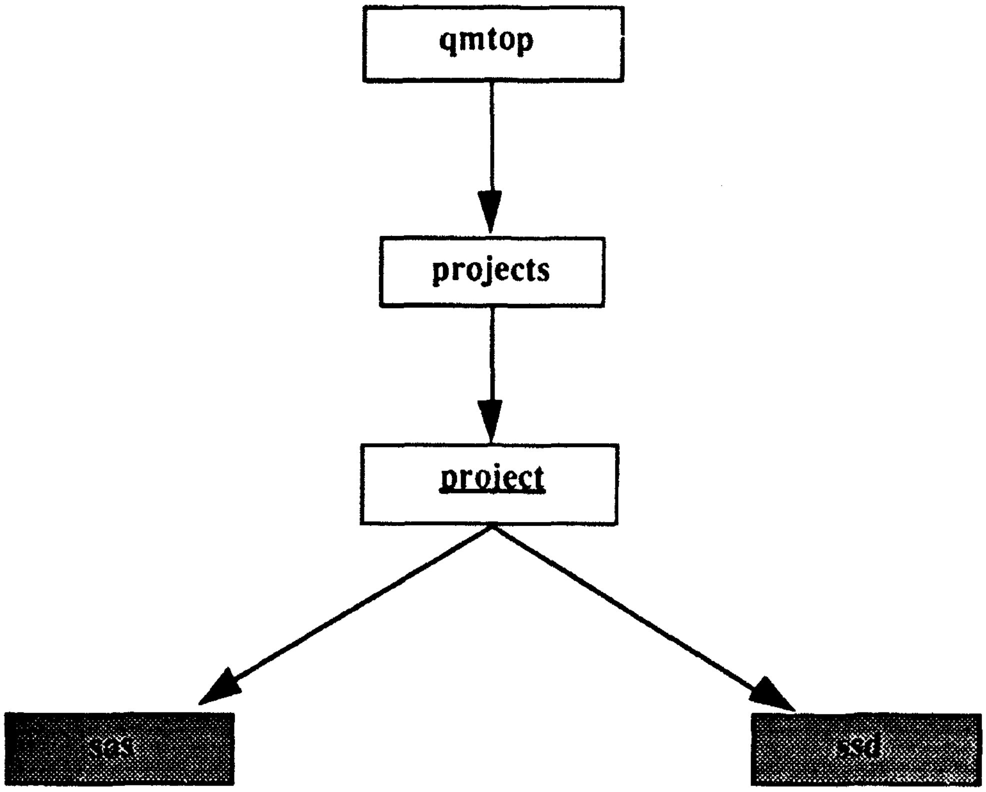

Figure 7-1. Directories Used in Subsystem-level Analysis Procedure

This command contains the name of a program that creates SAS data sets at the subsystem level. The input file to this program is database.ssd. The output files from this program are puds.ssd and gtots.ssd.

Step 4-4 Create subsystem level SAS file

This step takes the files created by the previous three steps and creates a subsystem level data set file.

The command to initiate this process follows:

```
sas sscmb.sas
```

This command contains the name of a program that creates a SAS data set that merges previously generated data sets. The output file from this program is sscmb.ssd.

SECTION 8

DESCRIPTION AND USE OF THE QUALITY PREDICTION MODELS

This section describes the quality prediction models and gives commands to invoke programs that apply the models using project data in files created by previous procedures in this report. The models described in this section were developed based on data provided by the Software Engineering Laboratory (SEL) of NASA's Goddard Space Flight Center. Data from four Ada projects consisting of 21 subsystems were used. Reference 1 provides detailed profiles of the data. These projects were concerned with the development of interactive, ground-based, scientific applications.

These models have had only limited validation using different projects. The application of the models should be viewed as a part of the process of validating the models in different environments. A different environment may lead to more or less defects than those predicted by the models. However, initial validation efforts have indicated that there may be a high correlation between actual and predicted defects, implying a linear relationship. The coefficient of this relationship must be determined externally from the models. When interpreting the model predictions for projects from substantially different development enviroments than those used to calibrate the model, the quality predictions may be interpreted as measures of relative merit in that environment.

The quality prediction models in this section may be classified according to the quality factor of interest, the time period over which a metric is predicted, and the level of granularity of the model. The models relate to the following quality factors: reliability, maintainability, and flexibility. For each of these quality factors, models are presented that predict metrics over two different time periods: (1) unit, system/integration, and acceptance (USA) testing, and (2) system/integration and acceptance (SA) testing. In the case of reliability, models are given at the subsystem level and the LUA level of granularity, while maintainability and flexibility models are presented only at the subsystem level.

The models that are presented represent a subset of the models that have been developed. These models were selected because they have produced the best results thus far in their category. For additional information concerning models that have been developed see references 1 and 4.

Prior to running any of the models, the user should enter the following command to change to the models directory:

```
cd /design1/qmtop/projects/project/sas/models
```

8.1 RELIABILITY MODELS

Four models are described: two each for predictions at the subsystem and LUA levels. Within each level, models are given at both the USA and SA time intervals. Note that, although these models are identified for convenience as reliability models, the research team has not had access to data on software-induced system failures. Instead, software defect data was analyzed to develop the models. Consequently, the models are more accurately identified as "reliability-related", because of the strong connection between defects and failures [6].

Reliability Model #1: Subsystem/USA

This model predicts the total number of defects per thousand lines of source code (TOTDEFSL) -- where "total" is used to mean defects reported during the activities of unit, system/integration, and acceptance testing. TOTDEFSL is predicted at the subsystem level.

The explanatory variables in the model are defined as follows:

Design Characteristic	Measure
Context Coupling	IMPEXP: number of unique declarations imported divided by the number of unique declarations exported
Visibility	CIMPIMP: number of unique cascaded declarations imported divided by the number of unique declarations imported

The basic form of the model is:

$$\log(Y) = a_0 + a_1 * \log(X_1) + a_2 * \log(X_2)$$

where

Y = dependent variable (TOTDEFSL)
X_i= independent (explanatory) variables
a_i= coefficients determined by multivariate regression

The calibrated model is as follows[1]:

$$\log(\text{TOTDEFSL}) = -0.04 + 0.51*\log(\text{IMPEXP}) + 0.26*\log(\text{CIMPIMP})$$

This model can be run using the following command:

```
sas rel_ss_usa.sas
```

The outputs from the model will be the predicted values of TOTDEFSL for all subsystems in the Ada system, where TOTDEFSL is the defects per thousand source lines of code reported during the activities of unit, system/integration, and acceptance testing.

Reliability Model #2: Subsystem/SA

This model is similar to Model #1, except it predicts the number of defects per thousand lines of source code that will occur during system/integration and acceptance testing only (SYACDEFSL).

The explanatory variables in the model are identical to those in model #1.

The calibrated model is [1]:

log (SYACDEFSL) = -1.42 + 0.70 * log (IMPEXP) + 0.46 * log (CIMPIMP)

This model can be run using the following command:

```
sas rel_ss_sa.sas
```

The outputs from the model will be the predicted values of SYACDEFSL for all subsystems in the Ada system, where SYACDEFSL is the defects per thousand source lines of code reported during the activities of system/integration, and acceptance testing.

Reliability Model #3: LUA/USA

This model predicts the probabilities that a library unit aggregation has 0, 1, 2, 3, 4, 5, or greater than 5 defects detected during the unit, system/integration and acceptance test activities [4]. From these probabilities, the expected total number of defects can be predicted at the LUA level and any other higher levels of aggregation (e.g., subsystem or project) by rolling up the LUA results.

The explanatory variables in the model are defined as follows:

Design Characteristic	Measure
Context Coupling	WITHS: number of library units "withed" per LUA
Functionality	VIS PROG UNITS: number of visible program units within the LUA

The model can be described as follows:

$p(X \leq i) = 1/(1 + \exp(\text{Intercept}_i - \text{model}), i = 0, ..., 5$

$p(X>5) = 1 - p(X \leq 5)$

where $p(X \leq i)$ is the probability that the number of defects, X, in the LUA is less than or equal to i (for i=0,1,...5), and $p(X > 5)$ is the probability that the number of defects is greater than five. The model is described in more detail in [4]. The model term is given by

$\text{model} = a_1 * (\text{WITHS}) + a_2 * (\text{VIS PROG UNITS})$

where the Intercept_i's and a_i's are the model parameter values as indicated in the table below:

Class	Identifier	Value	
	Intercept1	0.41	
	Intercept2	0.83	
	Intercept3	1.13	
	Intercept4	1.37	
	Intercept5	1.53	
	Intercept6	1.69	
Context Coupling	WITHS	0.07	(a_1)
Functionality	VIS PROG UNITS	0.0008	(a_2)

This model can be run by entering the following command:

```
sas rel_lu_usa.sas
```

Several processing steps occur when the command is executed. The direct outputs from the model are the probabilities of defects, as previously described. Using these probabilities, expected numbers of defects are computed for all LUAs and printed. Next, the expected defects at the LUA level are rolled up and printed at the subsystem level. Note that step 1-9 in Section 4 provides for the case in which there is a single "subsystem" that actually is the entire system.

Reliability Model #4: LUA/SA

This model is similar to Model #3, except that it predicts over the system/integration and acceptance test activities only [4]. The model parameters are as follows:

Class	Identifier	Value	
	Intercept1	0.60	
	Intercept2	1.03	
	Intercept3	1.35	
	Intercept4	1.54	
	Intercept5	1.74	
	Intercept6	1.89	
Context Coupling	WITHS	0.06	(a_1)
Functionality	VIS PROG UNITS	-0.007	(a_2)

This model can be run by entering the following command:

```
sas rel_lu_sa.sas
```

The outputs from the model are similar to model #3, except the predictions apply to the more restricted interval of system/integration and acceptance testing.

8.2 MAINTAINABILITY MODELS

Two models are described: the first predicts over the unit, system/integration, and acceptance (USA) testing activities and the second predicts over system/integration, and acceptance (SA) testing only. Both models predict, for subsystems, the probability that a defect in the subsystem will require less than 1 hour, less than 1 day, and less than three days of defect isolation effort [1].

Maintainability Model #1: Subsystem/USA

Model #1 is at the subsystem level, with coverage over unit, system/integration, and acceptance (USA) testing.

The explanatory variables in the model are defined as follows:

Design Characteristic	Measure
Context Coupling	WITHPLU: Mean number of library units "withed" per library unit aggregation
Visibility	VISHPUD: Percentage of hidden program unit declarations (i.e., number of hidden program unit declarations divided by number of hidden and visible program unit declarations)
Control Coupling	CALLPSUB: Mean number of subprogram invocation statements per subprogram in the subsystem

The model can be described as follows:

$$p(Y \leq i) = 1/(1 + \exp(\text{Intercept}_i - \text{model})),$$

such that

$p(Y \leq i)$ is the probability that a defect in the subsystem will require isolation effort less than or equal to category i, (for three categories: 1 hour, 1 day, and three days), and

$$\text{model} = a_1 * X_1 + a_2 * X_2 + a_3 * X_3$$

where the X_is represent the explanatory variables described above, and the Intercept_i's and a_i's are the model parameter values as indicated in the table below:

Class	Identifier	Value	
	Intercept1	0.147	
	Intercept2	2.244	
	Intercept3	3.525	
Context Coupling	WITHPLU	0.070	(a_1)
Visibility	VISHPUD	-1.301	(a_2)
Control Coupling	CALLPSUB	-0.029	(a_3)

This model can be run as follows:

```
sas mnt_ss_usa.sas
```

Any anomalies occurring while running the model will be noted in the SAS log file mnt_ss_usa.log. The output will be contained in the file mnt_ss_usa.lst. Five columns of output are produced. The first and second columns contain the name of the project and subsystem, respectively. The third, fourth and fifth columns contains the predicted probabilities that a effort to isolate a defect will be less than one hour, less than 1 day, and less than three days.

The outputs from the model will be the probabilities, for each subsystem, that the time to isolate defects in that subsystem will require less than 1 hour, less than 1 day, and less than three days.

Maintainability Model #2: Subsystem/SA

This maintainability model is similar to model #1 except that the prediction covers system/integration and acceptance (SA) testing only. The explanatory variables in the model are defined as follows:

Design Characteristic	Measure
Context Coupling	WITHPLU: Mean number of library units "withed" per library unit aggregation
Generality	GENS: Percentage of generic and instantiated library units in the subsystem
Control Coupling	CALLPEX: Mean number of subprogram invocation statements per executable unit in the subsystem
Locality	FINTPUD: Percentage of imported program unit declarations originating in the same subsystem as the importing unit

The model parameter values are indicated in the table below:

Class	Identifier	Value	
	Intercept1	-1.99	
	Intercept2	-0.15	
	Intercept3	1.33	
Context Coupling	WITHPLU	0.12	(a_1)
Generality	GENS	1.80	(a_2)
Control Coupling	CALLPEX	-0.03	(a_3)
Locality	FINTPUD	1.86	(a_4)

This model can be run as follows:

```
sas mnt_ss_sa.sas
```

Any anomalies occurring while running the model will be noted in the SAS log file mnt_ss_sa.log. The output will be contained in the file mnt_ss_sa.lst. Five columns of output are produced. The first and second columns contain the name of the project and subsystem, respectively. The third, fourth and fifth columns contains the predicted probabilities that a effort to isolate a defect will be less than one hour, less than 1 day, and less than three days.

8.3 FLEXIBILITY MODELS

Two flexibility models are described: the first predicts over the unit, system/integration, and acceptance (USA) testing activities and the second predicts over system/integration, and acceptance (SA) only. Both models predict, for subsystems, the probability that a non-defect change in the subsystem will require less than 1 hour, less than 1 day, and less than three days of isolation effort [1].

Flexibility Model #1: Subsystem/USA

Model #1 is at the subsystem level, with coverage over unit, system/integration, and acceptance (USA) testing. The explanatory variables in the model are defined as follows:

Design Characteristic	Measure
Context Coupling	PUDPLU: Mean number of imported program unit declarations per library unit aggregation
Generality	GENS: Percentage of generic and instantiated library units in the subsystem
Visibility	VISHPUD: Percentage of hidden program unit declarations (i.e., number of hidden declarations divided by number of hidden and visible program unit declarations)
Control Coupling	CALLPSUB: Mean number of subprogram invocation statements per subprogram in the subsystem

The model is similar to maintainability models in structure and results. The model parameter values are indicated in the table below:

Class	Identifier	Value	
	Intercept1	0.599	
	Intercept2	2.385	
	Intercept3	3.229	
Context Coupling	PUDPLU	0.015	(a_1)
Generality	GENS	2.462	(a_2)
Visibility	VISHPUD	-1.180	(a_3)
Control Coupling	CALLPSUB	-0.044	(a_4)

This model can be run as follows:

sas flx_ss_usa

Any anomalies occurring while running the model will be noted in the sas log file flx_ss_usa.log. The output will be contained in the file flx_ss_usa.lst. Five columns of output are produced. The first and second columns contain the name of the project and subsystem, respectively. The third, fourth and fifth columns contains the predicted probabilities that a effort to isolate a defect will be less than one hour, less than 1 day, and less than three days.

Flexibility Model #2: Subsystem/SA

This model is similar to flexibility model #1 except that the prediction covers system/integration and acceptance (SA) testing only. The explanatory variables in the model are defined as follows:

Design Characteristic	Measure
Context Coupling	PUDPLU: Mean number of imported program unit declarations per library unit aggregation
Generality	GENS: Percentage of generic and instantiated library units in the subsystem
Visibility	VISXPUD: Mean number of exported program unit declarations per library unit aggregation
Parameterization	PARVPUD: Mean number of parameters per visible program unit

The model parameter values are indicated in the table below:

Class	Identifier	Value	
	Intercept1	-2.50	
	Intercept2	-0.43	
	Intercept3	0.46	
Context Coupling	PUDPLU	0.03	(a_1)
Generality	GENS	3.03	(a_2)
Visibility	VISXPUD	0.33	(a_3)
Parameterization	PARVPUD	0.10	(a_4)

This model can be run as follows:

```
sas flx_ss_sa
```

Any anomalies occurring while running the model will be noted in the sas log file flx_ss_sa.log. The output will be contained in the file tlx_ss_sa.lst. Five columns of output are produced. The first and second columns contain the name of the project and subsystem, respectively. The third, fourth and fifth columns contains the predicted probabilities that a effort to isolate a defect will be less than one hour, less than 1 day, and less than three days.

LIST OF REFERENCES

1. Agresti, W. W., W. M. Evanco, M.C. Smith, D. R. Clarson, "An Approach to Software Quality Prediction from Ada Designs", MTR-90W00135, MITRE Corporation, September 1990.

2. Agresti, W. W., W. M. Evanco, M. C. Smith, "Early Experiences Building a Software Quality Prediction Model", Proceedings of the Fifteenth Annual Software Engineering Workshop, November 1990.

3. Doubleday, D. L., "ASAP: An Ada Static Source Code Analyzer Program", TR-1895, Department of Computer Science, University of Maryland, August 1987.

4. Evanco, W. M., and W. W. Agresti, "Statistical Representations and Analyses of Software", Proceedings of the 24th Symposium on the Interface of Computing Science and Statistics, College Station, Texas, 18-21 March 1992.

5. Evanco, W. M., W. M. Thomas, W. W. Agresti, "Estimating Ada System Size During Development", MTR-91W00132, MITRE Corporation, December 1991.

6. Musa, J. D., A. Iannino, and K. Okumoto, Software Reliability, New York: McGraw-Hill, 1987.

7. SAS Institute Inc., SAS Procedures Guide, Release 6.03 Edition. Cary, NC: SAS Institute Inc., 1988.

APPENDIX

EXAMPLE OF PROJECT SUMMARY REPORT

Project Summary Report
Project Counts Summary

Summary of ASAP Project Database: asap.db

UNIT NAME AND TYPE	SOURCE LINES TOTL	CODE	CMTS	BLNK	ADA STATEMENTS TOTL	EXEC	DECL	NESTDEPTH MAX	AVRGE	DECLARATIONS CON	OBJ	TYP	SBT	PAR	EXC	PACKAGE SP	BD	ST	SUBPROG SP	BD	ST	TASK SP	BD	ST	VIS WI	IN
ASAP_Reports.ProjectSummaryReport.BuildUnitCountsPart.AddToGlobalCounts (SB)	208	139	25	44	69	66	3	4	2.22	0	0	0	0	1	0	0	0	0	0	3	0	0	0	0	0	0
Parser.Apply_Actions (SB)	1773	317	1282	174	314	304	10	2	1.94	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	4	0
applutil1 (PB)	688	583	49	56	271	232	39	5	2.20	0	1	0	0	0	0	0	1	0	0	36	0	0	0	0	0	0
applutil1 (PS)	69	58	5	6	58	0	58	1	0.83	0	0	0	0	0	0	1	0	0	36	0	0	0	0	0	8	0
applutil2 (PB)	538	470	3	65	269	205	64	3	1.78	0	0	0	0	0	0	0	1	0	0	63	0	0	0	0	0	0
applutil2 (PS)	91	85	0	6	85	0	85	1	0.88	0	0	0	0	0	0	1	0	0	63	0	0	0	0	0	8	0
applutil3 (PB)	880	878	0	2	470	269	201	3	1.57	0	0	0	0	0	0	0	1	0	0	**	0	0	0	0	0	0
applutil3 (PS)	228	222	0	6	222	0	222	1	0.95	0	0	0	0	0	0	1	0	0	**	0	0	0	0	0	8	0
ASAP (SB)	358	241	39	78	125	85	40	5	1.79	1	19	0	2	7	1	0	0	0	0	3	2	0	0	0	11	1
asap_command_line_interface (PS)	58	15	31	12	13	0	13	1	0.15	0	0	0	0	6	1	1	0	0	1	0	0	0	0	0	6	0
asap_command_line_interface (PB)	472	246	116	110	159	116	43	7	2.96	0	22	0	0	24	0	0	1	0	5	6	0	0	0	0	6	0
Parser.ASAP_Reset (SB)	54	28	11	15	25	16	9	1	0.72	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	6	0
ASAP_Stack (PB)	149	65	31	53	37	20	17	3	1.65	1	4	0	0	6	0	0	1	0	0	9	0	0	0	0	1	0
ASAP_Stack (PS)	173	26	107	40	20	0	20	2	0.85	0	1	1	0	6	2	1	0	0	9	0	0	0	0	0	2	0
ASAP_Counts_Routines (PB)	124	69	14	41	38	22	16	4	1.71	0	1	0	0	9	0	0	1	0	0	10	0	0	0	0	3	0
ASAP_Counts_Routines (PS)	116	22	64	30	19	0	19	1	0.84	0	0	0	0	9	0	1	0	0	10	0	0	0	0	0	2	0
ASAP_Declarations (PS)	495	377	62	56	41	0	41	2	1.12	0	19	11	1	0	0	1	0	0	0	0	0	0	0	0	3	0
Parser.ASAP_Initialize (SB)	81	66	3	12	29	17	12	1	0.69	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	8	0
ASAP_Reports (PB)	253	155	39	59	91	48	43	5	2.03	5	7	1	2	17	0	0	1	0	0	6	9	0	0	0	4	2
ASAP_Reports (PS)	105	12	71	22	11	0	11	1	0.91	0	1	0	0	1	0	1	0	0	9	0	0	0	0	0	0	0
Binarytrees (PB)	235	83	58	94	48	33	15	3	2.08	0	3	0	0	14	0	0	1	0	0	9	0	0	0	0	1	1
BinaryTrees (PS)	225	48	70	107	23	0	23	2	1.13	0	0	4	0	13	0	1	0	0	7	0	0	0	0	0	1	1
Binary_Trees_Pkg (PB)	533	419	23	91	205	152	53	4	2.54	0	18	0	0	52	0	0	1	0	0	29	0	0	0	0	1	4
binary_trees_pkg (PS)	373	109	184	80	49	0	49	2	1.22	0	0	9	0	35	5	1	0	0	17	0	0	0	0	0	0	0
BlockStackPkg (PB)	70	40	7	23	23	11	12	3	1.70	1	4	0	0	0	0	0	1	0	0	5	0	0	0	0	0	0
BlockStackPkg (PS)	62	13	36	13	12	0	12	1	0.67	0	1	0	0	0	0	1	0	0	5	0	0	0	0	0	3	1
CaseInsensitiveLessThan (SB)	16	10	2	4	6	1	5	1	0.17	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	2	0
case_insensitive_string_comparison (PB)	174	119	11	44	55	35	20	4	1.96	1	8	0	0	18	0	0	1	0	0	12	0	0	0	0	0	0
case_insensitive_string_comparison (PS)	125	43	36	46	17	0	17	1	0.94	0	0	0	0	18	0	1	0	0	12	0	0	0	0	0	0	0
char_set (PS)	268	69	156	43	37	0	37	2	1.05	1	0	3	0	55	0	1	0	0	27	0	0	0	0	0	0	0
char_set (PB)	599	218	219	162	111	77	34	7	2.34	0	3	0	0	61	0	0	1	0	1	29	0	0	0	0	0	0
command_line (PS)	155	8	136	11	7	0	7	1	0.57	0	0	0	0	4	2	1	0	0	3	0	0	0	0	0	1	0
command_line (PB)	210	71	104	35	53	38	15	5	2.04	1	6	0	0	4	0	0	1	0	0	3	0	0	0	0	3	0
DatabasePkg (PB)	302	178	48	76	111	86	25	4	2.50	0	12	0	1	9	0	0	1	0	0	8	[illegible]	0	0	0	2	1
DatabasePkg (PS)	108	24	58	26	18	0	18	2	0.94	0	0	1	1	8	3	1	0	0	7	0		0	0	0	2	0
ASAP_Repr .s.ProjectSummaryReport.BuildUnitCountsPart.FinishGlobalCounts (SB)	159	117	15	27	49	48	1	3	1.96	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Parser.Apply_Actions.FinishUpUnit (SB)	131	78	27	26	50	38	12	4	2.34	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	4	0
Flow_Graph_Pkg (PB)	97	36	13	48	21	14	7	4	2.14	0	2	0	0	4	1	0	1	0	0	3	0	0	0	0	0	0
Flow_Graph_Pkg (PS)	74	25	13	36	15	0	15	2	0.87	0	0	1	0	4	0	1	0	0	3	0	0	0	0	0	3	2
ASAP_Reports.ProjectSummaryReport.BuildGlobalSummaryPart (SB)																										

ASAP REPORTS
Project Summary Report
Project Complexities Summary

Summary of ASAP Project Database: asap.db

UNIT NAME AND TYPE	MCABE CYCLO COMPL	UNIQUE OPND	UNIQUE OPTR	TOTAL OPRND	TOTAL OPRTR	POTENTIAL OPND	POTENTIAL OPTR	LENGTH PROG	LENGTH PRED	VOLUME PROGRM	VOLUME POTEN	LEVEL PROGRM	LEVEL LANG	PREDICTED EFFORT	PREDICTED TIME	BUGS
ASAP_Reports.ProjectSummaryReport.BuildUnitCountsPart.AddToGlobalCounts (SB)	12	56	24	519	569	53	6	1088	435	6878	347	0.0505	17.51	136311	2.104	2.3
Parser.Apply_Actions (SB)	301	314	321	319	946	6	4	1265	5277	11778	33	0.0028	0.09	4175874	64.442	3.9
applutil1 (PB)	62	135	72	1387	2002	113	72	3389	1400	26073	1393	0.0534	74.46	487914	7.530	8.7
applutil1 (PS)	0	15	43	29	194	0	72	223	292	1306	444	0.3401	151.07	3841	0.059	0.4
applutil2 (PB)	65	148	93	920	1634	123	126	2554	1675	20210	1982	0.0981	194.39	206063	3.180	6.7
applutil2 (PS)	0	15	70	29	302	0	126	331	488	2122	879	0.4144	364.31	5120	0.079	0.7
applutil3 (PB)	201	147	222	925	2258	132	400	3183	2789	27143	4817	0.1775	855.01	152933	2.360	9.0
applutil3 (PS)	0	15	207	29	850	0	400	879	1651	6851	3458	0.5047	******	13576	0.210	2.3
ASAP (SB)	36	134	73	528	747	56	10	1275	1399	9809	399	0.0407	16.22	241196	3.722	[illegible]
asap_command_line_interface (PS)	0	16	11	25	47	8	2	72	102	342	33	0.0970	3.22	3528	0.054	0.1
asap_command_line_interface (PB)	41	101	59	438	710	35	12	1148	1020	8406	261	0.0311	8.11	270636	4.17[illegible]	2.8
Parser.ASAP_Reset (SB)	1	26	18	52	91	17	2	143	197	781	81	0.1034	8.34	7552	0.117	0.3
ASAP_Stack (PB)	13	28	44	92	171	19	18	263	375	1623	193	0.1188	22.90	13661	0.211	0.5
ASAP_Stack (PS)	0	18	25	39	89	8	18	128	191	695	122	0.1760	21.50	3947	0.061	0.2
ASAP_Counts_Routines (PB)	17	38	40	144	234	31	18	378	412	2376	275	0.1158	31.86	20518	0.317	0.8
ASAP_Counts_Routines (PS)	0	15	22	35	107	10	18	142	157	740	135	0.1820	24.49	4065	0.063	0.2
ASAP_Declarations (PS)	0	380	14	433	448	7	0	881	3310	7596	20	0.0026	0.05	2936146	45.311	2.5
Parser.ASAP_Initialize (SB)	1	71	21	142	171	57	2	313	529	2042	347	0.1700	59.00	12013	0.185	0.7
ASAP_Reports (PB)	25	114	66	341	488	51	30	829	1178	6211	514	0.0827	42.46	75114	1.159	2.1
ASAP_Reports (PS)	0	5	16	6	43	3	18	49	76	215	92	0.4286	39.53	502	0.008	0.1
Binarytrees (PB)	15	28	44	145	260	25	20	405	375	2499	247	0.0989	24.44	25266	0.390	0.8
BinaryTrees (PS)	0	23	31	55	111	12	16	166	258	955	135	0.1409	18.97	6780	0.105	0.3
Binary_Trees_Pkg (PB)	79	57	79	533	1023	46	62	1556	830	11028	730	0.0662	48.26	166709	2.573	3.7
binary_trees_pkg (PS)	0	43	43	127	252	14	42	379	467	2436	325	0.1335	43.42	18240	0.281	0.8
BlockStackPkg (PB)	6	25	25	68	101	17	10	169	232	954	128	0.1346	17.28	7086	0.109	0.3
BlockStackPkg (PS)	0	13	17	23	44	6	10	67	118	329	64	0.1947	12.46	1689	0.026	0.1
CaseInsensitiveLessThan (SB)	1	8	13	17	27	6	2	44	72	193	24	0.1242	2.98	1556	0.024	0.1
case_insensitive_string_comparison (PB)	24	23	49	165	309	9	20	474	379	2925	141	0.0482	6.79	60710	0.937	1.0
case_insensitive_string_comparison (PS)	0	9	22	42	113	8	20	155	127	768	135	0.1753	23.60	4381	0.068	0.3
char_set (PS)	0	20	33	149	306	10	18	455	253	2606	135	0.0516	6.95	50461	0.779	0.9
char_set (PB)	49	25	54	396	706	16	22	1102	427	6947	199	0.0287	5.72	241988	3.734	2.3
command_line (PS)	0	10	15	14	34	6	6	48	92	223	43	0.1930	8.30	1155	0.018	0.1
command_line (PB)	14	40	39	182	244	19	6	426	419	2685	116	0.0432	5.02	62116	0.959	0.9
DatabasePkg (PB)	30	64	64	330	552	29	16	882	768	6174	247	0.0400	9.89	154242	2.380	2.1
DatabasePkg (PS)	0	21	27	40	86	8	14	126	221	704	98	0.1394	13.68	5048	0.078	0.2
ASAP_Reports.ProjectSummaryReport.BuildUnitCountsPart.FinishGlobalCounts (SB)	3	36	14	445	456	34	2	901	239	5085	186	0.0366	6.81	138936	2.144	1.7
Parser.Apply_Actions.FinishUpUnit (SB)	24	52	44	161	243	33	2	404	537	2660	180	0.0675	12.11	39423	0.608	0.9
Flow_Graph_Pkg (PB)	6	21	27	70	104	16	6	174	221	972	98	0.1010	9.90	9626	0.149	0.3
Flow_Graph_Pkg (PS)	0	19	19	42	61	8	6	103	161	541	53	0.0986	5.26	5481	0.085	0.2

Date : 09/10/92

ASAP REPORTS
Project Summary Report
Project Global Summary

Summary of ASAP Project Database: asap.db

GLOBAL MEASUREMENT	ALL UNITS	PKG SPECS	PKG BODIES	SUBPRG SPECS	SBPRG BODIES	TASK BODIES	PKG INSTANTS	SUB INSTANTS
UNITS ANALYZED	113	44	35	0	32	0	2	0
LINE COUNTS								
SOURCE LINES								
total	37112	13159	16838	0	7103	0	12	0
average/unit	328	299	481	0	221	0	6	0
maximum	2426	1928	2426	0	1773	0	7	0
LINES OF CODE								
total	18976	5805	9503	0	3659	0	9	0
average/unit	167	131	271	0	114	0	4	0
maximum	1272	865	1272	0	493	0	6	0
COMMENTS								
total	9650	4576	2966	0	2108	0	0	0
average/unit	85	104	84	0	65	0	0	0
maximum	1282	1034	646	0	1282	0	0	0
BLANK LINES								
total	8486	2778	4369	0	1336	0	3	0
average/unit	75	63	124	0	41	0	1	0
maximum	723	540	723	0	195	0	2	0
INLINE COMMENTS								
total	1133	626	420	0	87	0	0	0
average/unit	10	14	12	0	2	0	0	0
maximum	143	143	104	0	17	0	0	0
ADA STATEMENTS								
ALL STATEMENTS								
total	8662	1533	4784	0	2338	0	7	0
average/unit	76	34	136	0	73	0	3	0
maximum	598	222	598	0	369	0	4	0
EXECUTABLE ONLY								
total	5285	0	3299	0	1986	0	0	0
average/unit	46	0	94	0	62	0	0	0
maximum	452	0	452	0	350	0	0	0
DECLARATIVE ONLY								
total	3377	1533	1485	0	352	0	7	0
average/unit	29	34	42	0	11	0	3	0
maximum	222	222	201	0	40	0	4	0
UNITS WITHED IN								
total	232	101	68	0	58	0	5	0
average/unit	2	2	1	0	1	0	2	0
maximum	11	11	7	0	11	0	3	0
INSTANTIATIONS OF GENERICS								
total	49	13	25	0	9	0	2	0
average/unit	0	0	0	0	0	0	1	0
maximum	4	4	4	0	2	0	1	0
NUMBER OF DECLARED								

Summary of ASAP Project Database: asap.db

GLOBAL MEASUREMENT	ALL UNITS	PKG SPECS	PKG BODIES	SUBPRG SPECS	SBPRG BODIES	TASK BODIES	PKG INSTANTS	SUB INSTANTS
OBJECTS								
total	600	61	386	0	153	0	0	0
average/unit	5	1	11	0	4	0	0	0
maximum	65	19	65	0	31	0	0	0
FORMAL PARAMETERS								
total	1591	696	857	0	38	0	0	0
average/unit	14	15	24	0	1	0	0	0
maximum	166	144	166	0	9	0	0	0
EXCEPTIONS								
total	53	50	2	0	1	0	0	0
average/unit	0	1	0	0	0	0	0	0
maximum	11	11	1	0	1	0	0	0
COMPLEXITY MEASURES								
UNIT MAXIMUM NEST DEPTH								
average/unit	2	1	4	0	2	0	0	0
maximum	7	2	7	0	5	0	0	0
UNIT AVERAGE NEST DEPTH								
average/unit	1.3609	0.8263	1.9847	0.0000	1.4986	0.0000	0.0000	0.0000
maximum	3.6716	1.2245	3.6716	0.0000	3.1571	0.0000	0.0000	0.0000
CYCLOMATIC COMPLEXITY								
total	2211	0	1496	0	715	0	0	0
average/unit	19	0	42	0	22	0	0	0
maximum	301	0	201	0	301	0	0	0
SOME HALSTEAD MEASURES:								
PROGRAM LENGTH								
total	138443	70128	47634	0	20638	0	43	0
average/unit	1225	1593	1360	0	644	0	21	0
maximum	17131	17131	8669	0	2198	0	24	0
PROGRAM VOLUME								
total	1178038	643046	388531	0	146299	0	162	0
average/unit	10425	14614	11100	0	4571	0	81	0
maximum	205103	205103	96679	0	16378	0	94	0
PREDICTED EFFORT								
total	1173762279	1110035775	36928164	0	26791624	0	6716	0
average/unit	10387276	25228085	1055090	0	837238	0	3358	0
maximum	1106043885	1106043885	30788807	0	15302941	0	4396	0
PREDICTED TIME								
total	18113.6154	17130.1817	569.8791	0.0000	413.4510	0.0000	0.1036	0.0000
average/unit	160.2975	389.3223	16.2823	0.0000	12.9203	0.0000	0.0518	0.0000
maximum	17068.5785	17068.5785	475.1359	0.0000	236.1565	0.0000	0.0678	0.0000
PREDICTED BUGS								
total	392.6797	214.3491	129.5098	0.0000	48.7668	0.0000	0.0540	0.0000
average/unit	3.4750	4.8716	3.7003	0.0000	1.5240	0.0000	0.0270	0.0000
maximum	68.3677	68.3677	32.2264	0.0000	5.4593	0.0000	0.0313	0.0000

Project Summary Report
Project Global Summary

Summary of ASAP Project Database: asap.db

GLOBAL MEASUREMENT

USAGE OF ADA STATEMENTS ACROSS THE PROJECT

0	ABORT_STMT	0	ACCEPT_STMT	1191	ASSIGN
2775	CALL	0	CODE_STMT	45	CASE_STMT
0	DELAY_STMT	0	ENTRY_CALL	45	EXIT_STMT
0	GOTO_STMT	535	IF_STMT	22	LOOP_BASIC
130	LOOP_FOR	53	LOOP_WHILE	18	NULL_STMT
116	RAISE_STMT	355	RETURN_STMT	0	SELECT_STMT
15	ACCESS_TYPE_DECL	16	ARRAY_TYPE_DECL	9	BLOCK
105	COMPONENT_DECL	1	DERIVED_TYPE_DECL	0	ENTRY_DECL
17	ENUM_TYPE_DECL	47	EXCEPTION_DECL	192	FUNCTION_BODY
168	FUNCTION_SPEC	2	FUNCTION_STUB	3	GEN_FUNCTION_INSTANT
10	GEN_FUNCTION_PARAMETER_DECL	5	GEN_FUNCTION_SPEC	32	GEN_PACKAGE_INSTANT
6	GEN_PACKAGE_SPEC	0	GEN_PARAMETER_DECL	14	GEN_PROCEDURE_INSTANT
6	GEN_PROCEDURE_PARAMETER_DECL	4	GEN_PROCEDURE_SPEC	7	GEN_TYPE_PARAMETER_DECL
1	INT_TYPE_DECL	8	INCOMPLETE_TYPE_DECL	16	NUMBER_DECL
764	OBJECT_DECL	35	PACKAGE_BODY	38	PACKAGE_SPEC
0	PACKAGE_STUB	11	PRAGMA_STMT	14	PRIVATE_DECL
657	PROCEDURE_BODY	580	PROCEDURE_SPEC	28	PROCEDURE_STUB
0	REAL_TYPE_DECL	36	RECORD_TYPE_DECL	93	RENAMING_DECL
0	REPRESENTATION	82	SUBTYPE_DECL	0	TASK_BODY
0	TASK_SPEC	0	TASK_STUB	124	USE_CLAUSE
231	WITH_CLAUSE				

61% OF THESE ARE EXECUTABLE
39% OF THESE ARE DECLARATIVE

GLOSSARY

ASAP	Ada Source Analyzer Program
CU	Compilation Unit
DoD	Department of Defense
LU	Library Unit
LUA	Library Unit Aggregations
MOIE	Mission Oriented Investigation and Experimentation
PSR	Project Summary Report
SWEC	Software Engineering Center
SA	System and Acceptance Test
SAS	Statistical Analysis System
SEL	Software Engineering Lab
USA	Unit, System and Acceptance Test

©U.S. GOVERNMENT PRINTING OFFICE: 1993-710-[illegible]